4 AMIGA SONDERHEFT

Markt&Technik

DM 16,–
ÖS 120,– /Sfr 16,–
Lit 16000 /hfl 21,– /dkr 75,–

Super-Kurse

- C-Programmierung: Professionelles Arbeiten mit dem Amiga
- 68000-Assembler: Der leichte Einstieg

NEU: Workshop

- Erfolgreicher programmieren mit Power-Windows

Großer Vergleichstest

- Werkzeuge für Programmierer: Editoren, C-Compiler und Assembler

Zum Abtippen

- Profi-Tools für C-Compiler
- Trickreich: Universelle Laufschrift
- Komfortabler Sound-Editor

ASSEMBLER

C

ASSEMBLER

Alle Programme auf Diskette erhältlich

AMIGA SOFTWARE
Markt&Technik

SOFTWARE EXTRA

NEU: Amiga Extra Nr. 6: AudioWorx
Ihr privates Sampling-Studio. Das Digitalisieren von Sounds, sei es vom CD-Spieler, Tonband usw. ist mit AudioWorx ein Kinderspiel. Die Möglichkeit, aufgenommene Tracks zu verändern, sind vielfältig: Mit Hilfe zahlreicher Funktionen können Sie interessante Effekte erzeugen.
Bestell-Nr. 38748
DM 49,–* (sFr 45,–*/öS 490,–*)

NEU: Amiga Extra Nr. 9: Sonix
Licht aus - woommm. Den Amiga an den Hifi-Verstärker. Sonix in den Amiga. Monitor-Spot an - yeah! Die Klangdimensionen des Amiga haben Grenzen, aber die sind weit gesteckt. Wie weit, das erleben Sie, wenn Sie die Musik auf dieser Diskette abspielen lassen. 13 Stücke sind enthalten.
Bestell-Nr. 38753
DM 49,–* (sFr 45,–*/öS 490,–*)

Amiga Extra Nr. 5: Spiele
Spannende Unterhaltung mit vier Super-Spielen! Breaking out: Actionspiel mit schneller Grafik und tollem Sound. Decoder: Verwandeln Sie Ihren Amiga in eine Morsestation. Megamind: Anregende Unterhaltung für kluge Köpfe. Wikinger II: Spannendes Strategiespiel, angesiedelt im Mittelalter.
Bestell-Nr. 38752
DM 49,–* (sFr 44,–*/öS 490,–*)

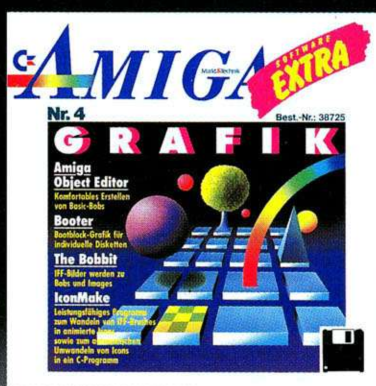






Amiga Extra Nr. 1: Grafik
CADos 3-D: Konstruktion und Rotation dreidimensionaler Körper. Funktionsplotter: Grafische Auswertung komplexer Funktionen. Fractal Construction Kit: Bilder einer fremden Welt.
Bestell-Nr. 38708
DM 49,–* (sFr 44,–*/öS 490,–*)

Amiga Extra Nr. 4: Grafik
Amiga Object Editor: The Bobbit. Iconmake. Booter.
Bestell-Nr. 38725
DM 49,–* (sFr 44,–*/öS 490,–*)



Amiga Extra Nr. 3: Spiele
Bliff: Eine ausgeklügelte Variante des Billards. Quadriga: Ein Spiel für Denker, angelehnt an »Vier Gewinnt«. Wikinger I: Ein Strategiespiel, angesiedelt im 10. Jahrhundert.
Bestell-Nr. 38724
DM 49,–* (sFr 44,–*/öS 490,–*)

Amiga Extra Nr. 2: Disk Utilities
DiskEd, Select Copy, DCopyII, Check, Bootgirl Plus.
Bestell-Nr. 38726
DM 49,–* (sFr 45,–*/öS 490,–*)

* Unverbindliche Preisempfehlung

Markt&Technik-Produkte erhalten Sie im Computerfachgeschäft, in den Fachabteilungen der Warenhäuser, im Versandhandel und in Ihrer Buchhandlung.



BESUCHEN SIE MARKT&TECHNIK AUF FOLGENDEN MESSEN:
Hannovermesse Industrie,
5.–12. April 1989, Halle 18, Stand B31/C32
Markt&Technik Verlag AG, Hans-Pinsel-Str. 2, 8013 Haar bei München

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0.
SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656.
ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0;
Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526;
Ueberreuter Media Verlagsges.mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.



Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

223/902

Wenn Sie dieses Sonderheft in den Händen halten, haben Sie wahrscheinlich die ersten Schritte mit dem Amiga bereits hinter sich. Die gewählten Themen helfen allen Programmierern, Probleme zu bewältigen.

- Der Assembler-Kurs erklärt die interne Arbeitsweise des Amiga. Gleichgültig, in welcher Programmiersprache Sie Projekte realisieren, diese wichtigen Kenntnisse bringen große Vorteile: Programme werden damit kürzer und schneller.

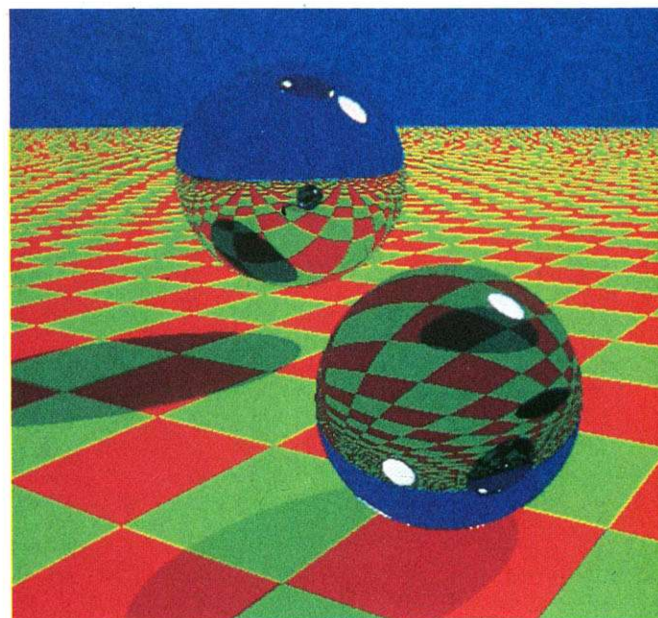


Die Freude am Experiment

- Es ist noch kein Meister vom Himmel gefallen. Diese Aussage bestätigt sich, wenn man den Amiga kennenlernt. Die Vorteile, die diese leistungsfähige Maschine bietet, sind erst nach einigen Experimenten intensiv zu nutzen. Das gilt ebenso für das Beherrschen komplexer kommerzieller Software wie für das Entwickeln trickreicher Programme.
- Vielleicht ist die Werbung von Commodore für die Amiga-Reihe daran mitschuldig, daß viele Käufer anfangs frustriert sind. Einfach ist es jedenfalls nicht, diesem vielseitigen Computer immer die richtigen Antworten zu entlocken. Vielleicht haben Sie anfangs innerlich geflucht, wenn Ihre Kommandos einfach nicht zum gewünschten Ziel führten. Mir ist es jedenfalls so gegangen, und es geht mir auch heute noch manchmal so. Wir bemühen uns, alle nötigen Grundlagen in den AMIGA-Sonderheften so einfach wie möglich zu erklären. Damit erhalten Sie die nötige Unterstützung, behalten den Spaß an Experimenten.
- Der umfangreiche C-Kurs zeigt, wie Sie die Systembibliotheken sinnvoll anwenden. Ein Grund mehr für jeden, sich diesen Kurs einmal genau anzusehen.
- Mit den zahlreichen Grundlagen, Beispiel-Listings und Tips in dieser Ausgabe werden Sie den Amiga noch besser einsetzen. Vielleicht veröffentlichen wir dann in einem der nächsten Sonderhefte Ihr Programm? Schicken Sie uns Ihre Vorschläge. Wir freuen uns auf viele Einsendungen.

Ihr
Ralf Sablowski
(Redakteur)

Ralf Sablowski





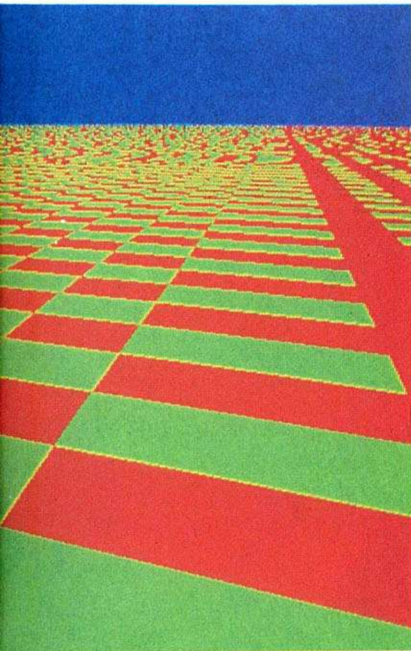
SEITE 24



Stellen Sie sich Ihre individuelle Werkzeugkiste zusammen

SEITE 6

INHALT

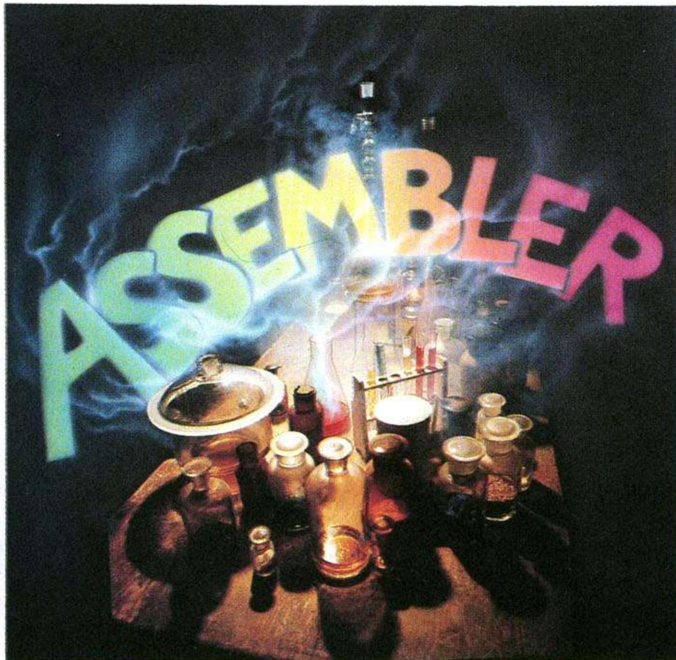


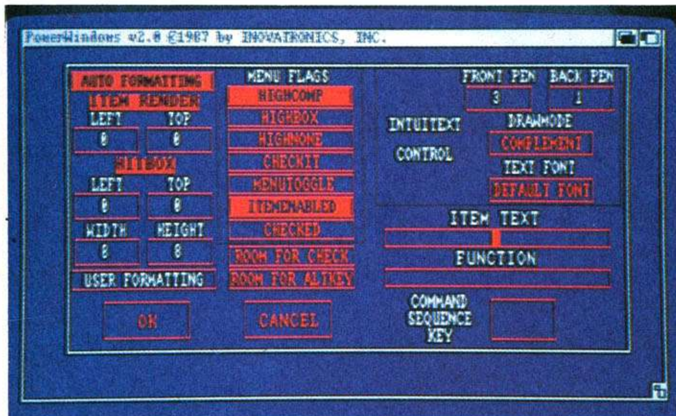
Fantastische 3D-Effekte erzielen Sie mit dem Ray Tracer

SEITE **132**

Assembler ist keine Alchimie. Schaffen Sie sich die nötigen Grundlagen für eine sinnvolle Anwendung dieser faszinierenden Sprache.

SEITE **74**





PowerWindows erleichtert die Schreibarbeit enorm

SEITE **101**

TOOLS

TIPS & TRICKS

SONSTIGES

Alle Programme aus Artikeln mit einem Disketten-Symbol finden Sie auch auf der Programmservice-Diskette zu diesem Sonderheft

WERKZEUGE FÜR

Ohne das richtige Werkzeug ist Programmieren unmöglich. Doch was gehört zur Ausstattung? Sie brauchen einen »Übersetzer«, der Ihre Gedanken in eine dem Amiga verständliche Form bringt. Wer sich die Arbeit zusätzlich erleichtern will, der benötigt einen komfortablen Editor und eine Programmier-Oberfläche als Ersatz für das CLI. Alle Werkzeuge stellen wir Ihnen auf den folgenden Seiten vor.



Egal, ob Sie Ihr Auto reparieren oder eine eigene Datenbank entwickeln – ohne das passende Werkzeug ist beides ein aussichtsloses Unterfangen. In den folgenden drei Übersichten finden Sie viele Dinge, die das Programmieren erleichtern sollen: Ohne den richtigen Compiler oder Assembler ist Programmieren undenkbar. Auf Seite 12 finden Sie einen Vergleich der beiden aktuellen C-Compiler von Lattice und Manx. Die wichtigsten Assembler stellen wir Ihnen auf Seite 19 vor.
Wer zum Programmieren einen eigenen Editor verwenden will, findet auf Seite 8 eine Gegenüberstellung einiger sehr komfortabler »Schreibwerkzeuge«. Wenn Sie sich nicht in unnötiger Tipperei bei den einzelnen Programm-Aufrufen verheddern wollen, sollten Sie auf eine gute Benutzeroberfläche nicht verzichten. Das CLI bietet hier nur wenig Hilfe. Eine gute Alternative ist die Shell auf Seite 115. Für Mausfans haben wir je eine Benutzeroberfläche für den Lattice- und den Manx-Compiler (Seite 121 und 124). Sie werden feststellen, daß mit diesen Werkzeugen Ihre Programme sich zwar immer noch nicht von selbst schreiben. Dafür wird Ihre Arbeit aber um einiges erleichtert.

1. EDITOREN

Editoren sind Schreibwerkzeuge. Beim Programmieren werden sie verwendet, um den sogenannten Quelltext eines Programms einzugeben. Im Unterschied zur Textverarbeitung werden hier keine Seiten- oder Druckformatierungen benötigt. Stattdessen ist es hilfreich, wenn Sie den Editor mit verschiedenen Tastenkombinationen belegen können oder wenn verschiedene Quelltexte gleichzeitig zu bearbeiten sind.

Andere hilfreiche »Beigaben« eines Editors sind Such- und Ersetzfunktionen, einfaches und schnelles Scrollen und komfortable Blockoperationen. Eine Übersicht finden Sie auf Seite 8. Sie stellt die vier wichtigsten Editoren ausführlich vor.

2. COMPILER

Der Computer versteht nur Zahlenkombinationen aus Null und Eins – die sogenannte Maschinensprache. Wir sind es aber gewohnt, in anderen Ausdrücken zu denken als »01001011101«. Um die gewohnte Denkweise des Menschen zu unterstützen, gibt es sogenannte »Hochsprachen«. Diese bieten mit Ausdrücken wie »if« oder »while« eine enge Anlehnung an die englische Sprache. Um diese Hochsprache in die Maschinensprache zu übersetzen, benötigen Sie den Compiler. Für den Amiga gibt es inzwischen viele Programmiersprachen. Wir beschränken uns jedoch in diesem Heft auf C. Auf Seite 12 finden Sie einen Überblick über die C-Compiler von Manx und Lattice.

PROGRAMMIERER



3. ASSEMBLER

Neben den Hochsprachen gibt es für jeden Mikroprozessor eine Sprache, die diesem speziell angepaßt ist. Diese »Assembler-Sprache« besteht aus kurzen Befehlsworten, die eine Aktion des Prozessors beschreiben. Auch diese muß jedoch in Maschinensprache übersetzt werden. Mit einem Assembler können Sie Programme schreiben, die die Fähigkeiten des Prozessors optimal ausnutzen und die daher sehr kurz und extrem schnell sein können. Der Nachteil ist allerdings die – zumindest für den Ungeübten – schlechtere Lesbarkeit des Quellcodes.

Sie finden auf Seite 19 eine Übersicht der wichtigsten Assembler, die für den Amiga erhältlich sind.



AmigaDOS

...wenn es ums Editieren von Texten geht. Immer mehr leistungsfähige Alternativen drängen auf den Markt. Wir haben die vier interessantesten verglichen.

Die Zeiten des guten alten »ED« aus dem C-Directory jeder Systemdiskette sind endgültig vorbei. Für ernsthaftes Arbeiten benötigt man vernünftiges Werkzeug – auch beim Programmieren. Und da steht der Editor als Hilfsmittel zur Eingabe des Quelltextes buchstäblich an erster Stelle. Manche Sprachen (so auch AmigaBasic) haben zwar einen eigenen Editor eingebaut, jedoch stellen diese meistens nur die allernötigsten Funktionen zur Verfügung, und das ist für komfortables Arbeiten einfach nicht genug.

Der Unterschied zwischen einem Editor und einem Textverarbeitungs-Programm besteht grundsätzlich darin, daß ein Editor reinen ASCII-Text bearbeitet und abspeichert. Die Textverarbeitung dagegen verwendet spezielle Steuerzeichen für die Formatierung, für Schriftattribute (beispielsweise fett, unterstrichen etc.) und andere Extras. Doch dies alles benötigen Sie normalerweise nicht im Quelltext Ihres Programms, dort wird man niemals Unterstreichungen oder Blocksatz finden. Man könnte also sagen, Editoren sind Textverarbeitungen, denen einfach ein paar Funktionen fehlen. Wieviel tatsächlich fehlt, ob wichtige oder überflüssige Funktionen vorhanden sind, werden unsere vier Vergleichskandidaten zu zeigen haben:

- »ProEd« von Digital Projects,
- »SuperEd« von Markt&Technik,
- »CygnusEd Professional« von ASDG sowie
- »EdWork« von Data Becker.

Bevor wir uns in die Details stürzen, wollen wir erst einmal sehen, was alle Kandidaten gemeinsam zu bieten haben. Da ist als erstes und auffälligstes die Menüführung. Fast alle Befehle und Funktionen sind über Menü aufrufbar und erleichtern somit dem Einsteiger die Handhabung. Der Fortgeschrittene kann aber trotzdem die meisten Menüs auch durch schnelle Tastendrücke ersetzen. Außerdem zeigen sich alle vier Editoren in Sachen Blockbefehle deutlich besser gerüstet als der alte ED, und sie sind auch allesamt um einiges schneller. Allerdings gibt es gerade in diesem wichtigen Punkt oft deutliche Unterschiede zwischen den einzelnen Programmen.

Amateur mit Ambitionen

Beim ProEd von Digital Projects (Bild 1) fällt schon beim Laden ein angenehmer Service auf: Man bekommt das Programm in zwei Versionen, einmal mit englischer und einmal mit deutscher Benutzerführung. Nach dem Start erscheint dann das ProEd-Fenster auf dem Workbench-Screen.

Der Textspeicher wird wie bei den meisten Editoren dynamisch verwaltet, folglich ist die Textlänge lediglich durch die Größe des vorhandenen freien Speichers begrenzt. Alle Funktionen sind über Menü und Tastatur abrufbar, beim Aufruf des Programms vom CLI ist aber auch die Übergabe von Parametern wie Seitenlänge, Zeilenlänge, Wordwrap-Funktion und so weiter möglich.

Am rechten und unteren Bildschirmrand befinden sich Schiebe-Gadgets, mit deren Hilfe man schnell einen bestimmten Textausschnitt in das Fenster schieben kann. Beim Scrolling des Textes über die Cursortasten fällt allerdings unangenehm auf, daß eine unschöne Eigenart des AmigaDos-Ed übernommen wurde: Der Text wird nicht zeilenweise verschoben, sondern es wird jedesmal das Fenster gelöscht und der Text ab der entsprechenden Position neu ausgegeben.

Angenehm ist dagegen das Arbeiten in Textblöcken. ProEd verfügt über je zwei unabhängige Block- und Zeilen-Puffer, wodurch das Umkopieren oder Verschieben von Textausschnitten komfortabel ist. Es ist lediglich schade, daß Textblöcke nur mit den Cursortasten und nicht mit der Maus markiert werden können. Dafür kann man aber über jeweils einen Menüpunkt beziehungsweise Tastendruck den gesamten Text, das Wort unter dem zu 26 solcher Makro-Definitionen werden angenommen. Im Grunde genommen ist dieser Makro-Modus also nichts anderes als eine leistungsfähige Floskeltasten-Belegung.

Ansonsten bietet der ProEd wenig Besonderheiten, abgesehen von der Speicherung des Textes als »Stream« (ohne »Carriage-Returns« und »Linefeeds«, zur Übernahme des Textes in eine Textverarbeitung) und der Option, beim Speichern von Texten automatisch ein Sicherheits-Backup erstellen zu lassen.

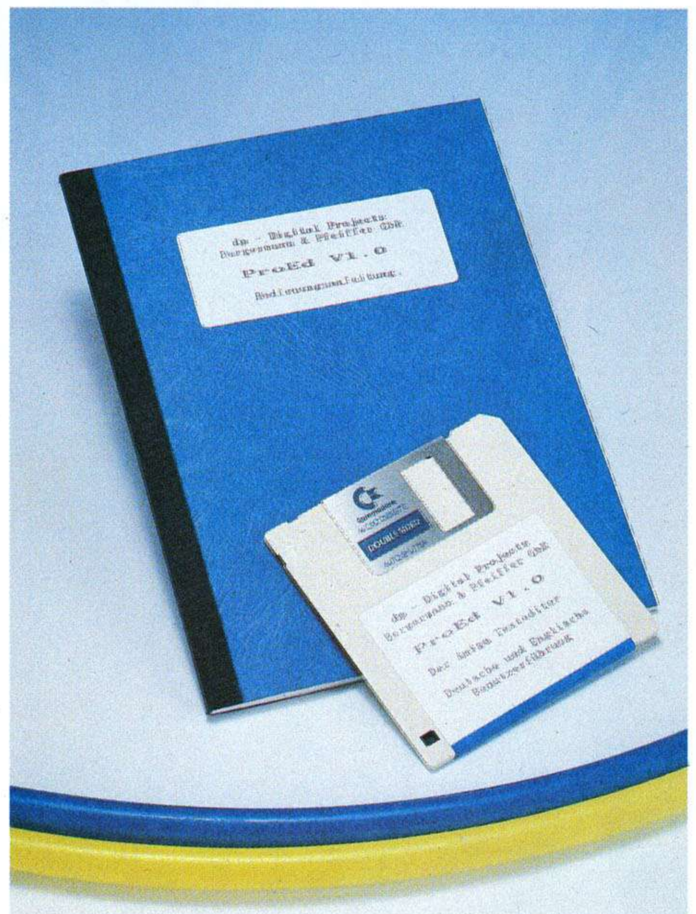


Bild 1. Das Handbuch zu ProEd ist eher spartanisch

Cursor, die Cursorzeile und ähnliches schnell markieren.

Vom Namen her etwas irreführend ist der Makro-Modus. Hier werden nicht wie bei anderen Editoren Befehlssequenzen auf eine Taste gelegt, sondern lediglich Textausschnitte. Wenn man sich nämlich nach der Markierung eines solchen Ausschnittes in den Makro-Modus begibt, legt man den Text auf eine bestimmte Taste. Ab jetzt kann immer im Makro-Modus dieser Text durch Druck auf diese Taste eingefügt werden. Bis

Von der Geschwindigkeit her läßt das Programm zu wünschen übrig. Im Durchschnitt erwies es sich als langsamster Editor in diesem Test. Lediglich beim Text-Ersetzen lag ProEd noch vor den beiden Extremen CygnusEd und EdWork.

Das Handbuch mit 25 Seiten macht einen recht verletzlichen Eindruck, häufiges Blättern wird wohl unweigerlich Zerfallserscheinungen nach sich ziehen. Der Text ist auf das absolute Mindestmaß reduziert und wenig übersichtlich. Auch die

hat ausgedient...



Bild 2. Kleine Verpackung, große Funktionsvielfalt: SuperEd

geradezu amateurhafte äußere Form (jedes Blatt nur einseitig bedruckt, keinerlei Hervorhebungen wie Fettschrift oder Unterstreichung, Tabellen mit normalen ASCII-Zeichen gebastelt) sowie die zahlreichen Rechtschreib- und Grammatikfehler unterstützen diesen schlechten Eindruck.

ProEd von Digital Projects gehört mit einem Preis von 58 Mark in die Sparte Billig-Software, kann aber von der Leistungsfähigkeit her nicht mit den Konkurrenten mithalten. Es bleibt lediglich noch die Frage, wie die Entwickler es geschafft haben, für ein solches Programm ungefähr doppelt so viel Speicher zu belegen (92 KByte) wie der direkte Mitbewerber SuperEd (45 KByte).

Ungefähr auf dem gleichen Preisniveau wie der ProEd liegt »SuperEd«, von Markt & Technik (Bild 2). Dieses Programm öffnet seinen eigenen Screen, dadurch wird der ganze 80-Zeichen-Bildschirm ausgenutzt. SuperEd überrascht im ersten Moment durch fehlende Schiebe-Gadgets. Man wird jedoch mehr als entschädigt, wenn man mit gedrückter linker Maustaste an den unteren Bildschirmrand fährt. Der Text reagiert, indem er nach oben wegscrollt. Dasselbe gilt umgekehrt für den oberen Rand – eine sehr angenehme Scroll-Variante, die noch dazu eine rasante Geschwindigkeit an den Tag legt.

SuperEd bietet zwei Block- und einen Zeilen-Puffer, mit denen die üblichen Operationen ausgeführt werden. Die Markierung von Textausschnitten funktioniert sehr angenehm. Ein einzelnes Wort wird durch zweimaliges Anklicken markiert, ein Block dagegen, indem man bei gedrückter rechter Maustaste den Bereich mit der Maus überstreicht. Gegebenenfalls scrollt der Text auch automatisch weiter, wenn der Mauszeiger dabei an einen Bildschirmrand stößt. Im Zusammenhang damit recht interessant ist auch die Undo-Funktion. Damit werden unbeabsichtigte Änderungen innerhalb einer Zeile wieder rückgängig gemacht.

Die Ausgabe des Textes auf einen Drucker wurde besser als in mancher Textverarbeitung gestaltet. SuperEd startet einen sogenannten Druckerspooler als eigenständigen Task. Dies ist ein eigenes Programm, das neben dem eigentlichen Editor läuft und den Text an den Drucker weitergibt. Dadurch ist der Editor sofort wieder zur Arbeit bereit, die Druckerausgabe läuft unabhängig weiter. Soweit gewünscht, wird als Textkopf der Name der Datei sowie das aktuelle Datum automatisch mitausgegeben. Auch erfolgt der Druck auf Wunsch in den Standard-Schriftarten NLQ, Pica, Elite oder Fine.

Die Funktionen Upper-, Lower- und Switch-Case ändern in einem markierten Textausschnitt entweder alle Groß- in Kleinbuchstaben, alle Klein- in Großbuchstaben, oder sogar beides gleichzeitig. Normalerweise besteht ein Quelltext lediglich aus Text. Doch in Ausnahmefällen ist auch die Eingabe von beliebigen ASCII-Zeichen praktisch oder gar nötig. SuperEd ermöglicht die Eingabe solcher Sondercodes in Form von Hexadezimal-Zahlen.

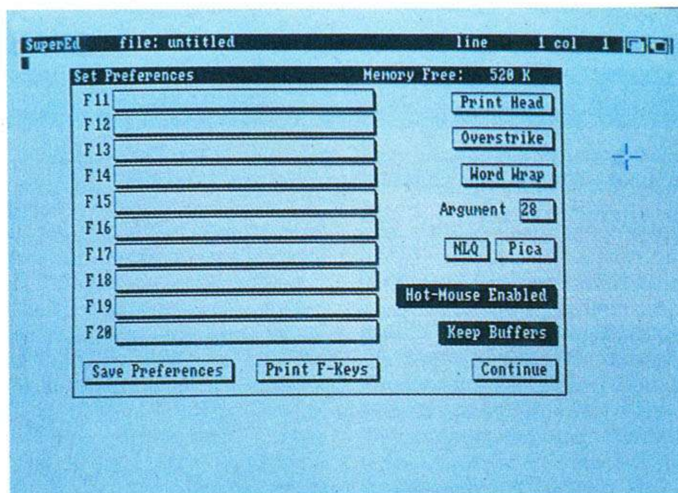


Bild 3. SuperEd wartet mit vielen Funktionen auf

Für die Aufnahme von Floskeltexten stehen die Funktionstasten zusammen mit der Shift-Taste zur Verfügung. Es stehen also maximal zehn verschiedene Floskeltexte bereit.

Wer während des Editierens einmal eine Datei auf Diskette löschen oder einen DOS-Befehl ausführen will, der braucht den Editor nicht zu verlassen, sondern kann ein CLI-Fenster öffnen und nach Gebrauch wieder schließen.

Für umfangreichere Aktionen neben dem Editieren aber hat SuperEd noch eine praktische Variante zu bieten: die sogenannte »Hot-Mouse«. Schaltet man diese Option ein (Bild 3) und beendet das Programm, so

Meisterhaft

wird der Task nicht aus dem Speicher entfernt, sondern bleibt aktiv. Nun kann man zum Beispiel den gerade geschriebenen Quelltext compilieren. Um nun wieder in den Editor zu gelangen, genügt es, beide Maustasten gleichzeitig zu drücken, und schon ist der Editor wieder da, auf Wunsch auch mit dem alten Text.

Insgesamt arbeitet der SuperEd ziemlich schnell. Lediglich aufgrund des »Wordwrappings« kann es zu unangenehmen Verzögerungen kommen. Paßt nämlich das eben geschriebene Wort nicht mehr in die aktuelle Zeile, so wird es in die nächste umkopiert. Dies dauert allerdings so lange, daß während des Kopierens bei schnellem Schreiben regelmäßig Tastendrücke ignoriert werden – ein Schönheitsfehler, der bei längeren Fließtexten die Nerven des Benutzers doch erheblich belastet.

Das »Handheft« umfaßt 35 Seiten, ist übersichtlich (wenn auch nicht allzu ausführlich) und leicht verständlich. Somit erhält man für den Preis von 39 Mark ein schnelles und hilfreiches Werkzeug. SuperEd ist das preiswerteste und mit 45 KByte auch kürzeste Programm in diesem Test.

Der »CygnusEd« von ASDG (Bild 4) kann mit gutem Recht als Sensation bezeichnet werden. Er bietet eine unübertroffene Vielzahl von durchdachten und hilfreichen Befehlen und Funktionen. Zudem ist er über die Sprache ARexx bei Bedarf programmier- und erweiterbar.

Schon beim Nachladen einer Datei im »File-Selector« (Bild 5) fällt die neue Scroll-Routine auf. Wo bei allen anderen Pro-

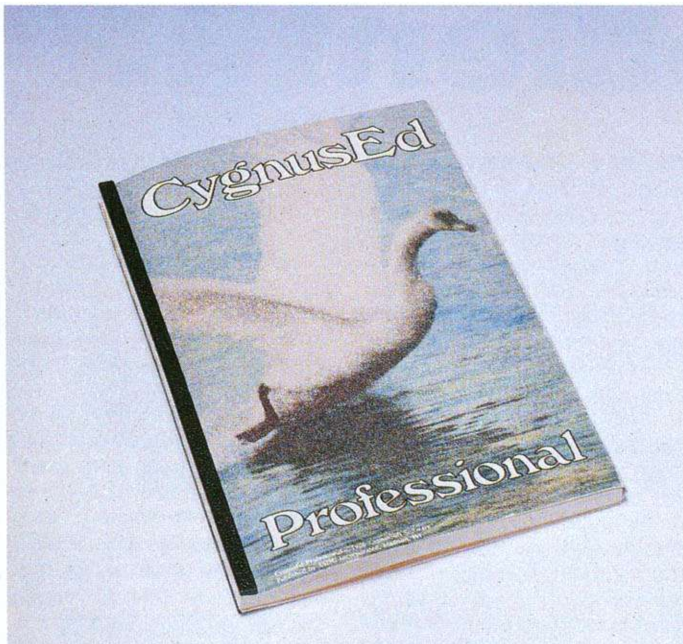


Bild 4. Eine Meisterleistung ist der CygnusEd von ASDG

grammen der Text zeilenweise verschoben wird, geschieht das bei CygnusEd punktweise und beliebig fein. Die damit erreichte Übersichtlichkeit und Geschwindigkeit sind begeisternd. Es ist wahrscheinlich unmöglich, alle Vorteile des CygnusEd gegenüber anderen Editoren aufzuzählen, wir beschränken uns auf die interessantesten Highlights.

Zu diesen gehört zum Beispiel die Option, mehrere Texte in verschiedenen Fenstern unabhängig voneinander zu bearbeiten. Selbstverständlich können Sie dabei Blöcke zwischen den verschiedenen Fenstern hin- und herkopieren. Diese zusätzlichen Fenster können beliebige Ausmaße annehmen. Als Basis können alle übergroßen Formate (Overscan) bis zu 704 x 584 Punkten (in Interlace) eingestellt werden. Auch beliebig große Screens sind möglich (beispielsweise mit dem PD-Programm »VScreenHandler«), allerdings sind dann natürlich nur Ausschnitte am Monitor erkennbar. Auf Wunsch kann der Editor seine Fenster im Workbench-Screen oder in einem eigenen Screen öffnen.

Wer den Editor als zweiten oder höheren Task laufen läßt, gibt im Menü die gewünschte Task-Nummer ein, um die verfügbare Rechenzeit auf die verschiedenen Programme aufzuteilen.

Hat man einige Zeit (genauer gesagt: 15 Minuten) mit dem CygnusEd gearbeitet, so macht sich eine ganz besondere Funktion bemerkbar: das Autosave. Das Programm öffnet einen Requester, teilt Ihnen die vergangene Zeit seit dem letzten Abspeichern (voreingestellt sind 15 Minuten) sowie die Zahl der seither vorgenommenen Veränderungen mit. Auf Wunsch wird nun der Text abgespeichert. Der Timer für Autosave kann über Menü gestellt beziehungsweise ausgeschaltet werden. Das sogenannte »Safe Saving« verwendet eine Temporärdatei, um Datenverlusten während des Überschreibens der alten Datei vorzubeugen.

CygnusEd verfügt wie SuperEd über eine Hot-Start-Funktion. Das Programm, welches nach dem Beenden noch im Speicher steht, wird hierbei mit einer Tastenkombination wieder gestartet. Sie können diesen Task auch mit einer anderen Tastenkombination entfernen (beispielsweise wenn beim Compilieren der Speicher knapp wird). Die gesamten Voreinstellungen von Autostart, Autosave, Bildschirmgröße, Task und vieles mehr lassen sich auf Wunsch auch gesammelt abspeichern und bei Bedarf neu laden, womit man wertvolle Zeit spart.

Das Programm bietet auch echte Makros, mit denen man beliebige Befehle und Funktionen durch lediglich eine Tastenkombination aufrufen kann. Natürlich lassen sich auch diese Makros abspeichern. Mit Makros verwandt ist der Befehl »Repeat key/menu«, womit sich ein Menü- oder Tastaturbefehl bis zu einer Milliarde mal automatisch wiederholen läßt.

Beim Programmieren von langen C-Programmen oder auch komplizierten Formel-Ausdrücken verirrt man sich oft in geschachtelten Klammerebenen. CygnusEd sucht automatisch zu einer geöffneten Klammer die entsprechende geschlossene und andersherum. Bei langen Quelltexten sind die Marken praktisch – eine Art zweidimensionaler Tabulator. Bis zu drei Stellen im Text



Bild 5. Der File-Selector von CygnusEd ist ein Beispiel für sauber durchdachte Hilfestellung für den Benutzer

können markiert und dann direkt angesprungen werden.

Leider bietet CygnusEd lediglich einen einzigen Block-Puffer, der aber auf jede erdenkliche Art manipulierbar ist – bei der sonstigen Vielfalt an ausgeklügelten Funktionen geradezu erstaunlich. Suchen und Ersetzen dagegen wird nicht nur vorwärts oder rückwärts, sondern bei Bedarf auch mit Wildcards, also Platzhaltern, durchgeführt.

Der Ausdruck von Texten oder Textblöcken wird beim CygnusEd wie beim SuperEd von einem Spooler mit eigenem Task übernommen (siehe oben). Dieser bietet zudem noch die Option, aus dem Quelltext Tabulatoren in Leerzeichen und andersherum zu verwandeln. Somit gibt es keine Probleme mit der Druckeranpassung.

Wem alle diese Funktionen immer noch nicht genug sind, der kann CygnusEd auch noch programmieren. Dazu ist allerdings ein Interpreter der Sprache »ARexx« vonnöten, welche in Amerika (dem Herkunftsland von CygnusEd) sich mehr und mehr zum Standard entwickelt. Mit ihrer Hilfe kann der anspruchsvolle Benutzer noch eigene Formatierungsroutinen oder andere Textbearbeitungen programmieren. Der ARexx-Interpreter kostet 50 Dollar.

Im Geschwindigkeitsvergleich ist CygnusEd der schnellste Editor (wie schon erwähnt auch beim Scrollen), mit der Ausnahme der Ersetz-Funktion. Diese zeigt nämlich jede vorgenommene Änderung sofort am Bildschirm an, womit natürlich viel Zeit verbraucht wird. Auf den ersten Blick mag das zwar wegen der Ausführungszeit als Nachteil erscheinen, für den Programmierer bietet es aber eine einfache Kontrolle über alle Änderungen. Auch wenn bei der Verwendung von Wildcards versehentliche Änderungen vorgenommen wurden, erkennt man diese sofort.

Das Handbuch ist erfreulich solide, sowohl vom Inhalt als auch vom Material her. Es ist übersichtlich und erklärt allgemeine Benutzerhinweise sowie kompliziertere Besonderheiten ausführlich in gesonderten Kapiteln. Ein Nachteil mag die englische Sprache sein, das

Ausbau eingebaut

Handbuch wurde nämlich bis jetzt noch nicht übersetzt.

Bei einem Preis von 198 Mark liegt dieser Editor aber auch in der gehobenen Preisklasse. Doch für Programmierer ist dieses Programm eine leistungsfähige Hilfe und ganz sicher sein Geld wert.

Haben Sie mit Ihrem Editor schon einmal »Breakout« oder »TicTacToe« gespielt? Mit »Edwork« von Data Becker (Bild 6) ist dies kein Problem, wenn der erste Eindruck dies auch nicht vermuten läßt. Wer EdWork nämlich einfach so aufruft, der findet sich in einem Programm mit dem Bedienungskomfort des AmigaDOS-Ed. Es sind keine Menüs installiert, und die Befehle werden über Escape-Sequenzen aufgerufen.

Der Grund dafür: Dieses Programm ist weniger ein Editor als vielmehr ein Sprach-Compiler mit einigen Fertigteilen – sozusagen ein »Bausatz«. Die dazugehörige Sprache heißt EPL (Editor Programming Language). Damit kann der Bediener Menüs erstellen, Funktionen zusammenbasteln und Tasten beliebig belegen. Dazu wird noch ein Compiler geliefert, der

EPL in Maschinencode übersetzt und so die Ausführungsgeschwindigkeit wesentlich erhöht.

Damit der Benutzer nun nicht gleich programmieren oder das EPL-Programm auf einem tastengesteuerten Editor schreiben muß, werden schon einige fertige Compilate mitgeliefert. Darunter befinden sich Anpassungen, die EdWork bedienungskompatibel zu Profimat, Textomat, TxED oder dem AmigaDOS-Ed machen (Bild 7).

Außerdem mitgeliefert werden Demoprogramme, die Spezialitäten und Leistungsfähigkeit des EdWork demonstrieren, so zum Beispiel die beiden oben genannten Spiele. Praktische Anwendungen sind zum Beispiel der C- und Assembler-»Beautyfier«. Die beiden Programme formatieren Quelltexte, um eine bessere Lesbarkeit zu erreichen. Auch eine Sortier-

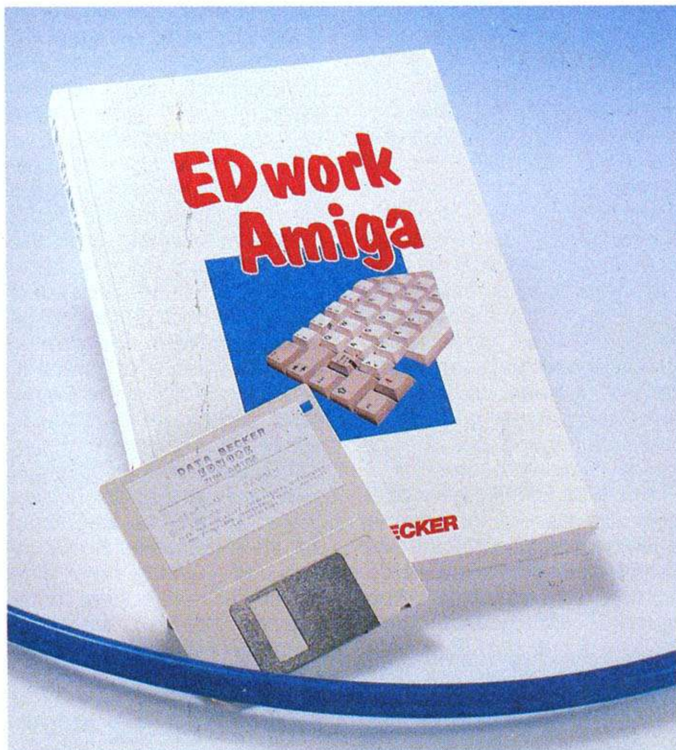


Bild 6. Das Handbuch zu EdWork ist eine ergiebige Informationsquelle für Einsteiger wie für Profis

routine ist auf der Diskette zu finden.

Wer aber einen wirklich guten und einigermaßen komfortablen Editor haben möchte, der kommt nicht darum, sich durch die über 120 Befehle von EPL zu kämpfen. Hier sind insgesamt bis zu 65534 Makros frei definierbar, was für den Anfang wohl reichen wird.

Eine der herausragenden Besonderheiten, welche mit EdWork realisierbar sind, ist das sogenannte »Folding«. Dabei faltet man unwichtige Textbereiche einfach weg, sie werden also unsichtbar, sind aber immer noch im Textspeicher vorhanden. Bei Bedarf faltet man sie dann wieder auf. Wer zum Beispiel die am Anfang fast jedes C-Programms stehenden Standard-Strukturen für Screen, Window und so weiter wegfaltet, verkürzt den Quelltext optisch und sorgt so für Übersichtlichkeit.

Wie jede Standard-Programmiersprache bietet EPL die Verwendung von Variablen, Grundrechenarten und Vergleichsoperatoren. Auch Schleifen und bedingte Ausführung sind implementiert. Und dazu eine große Zahl editorspezifischer Befehle.

Für Blockmanipulationen stehen neben den Befehlen das sogenannte Folding zur Verfügung. Außerdem hantiert man mit mehreren Blöcken gleichzeitig, indem man die Blockdaten einfach auf einen entsprechenden Stack auslagert und später wieder holt. Natürlich umfaßt EPL auch alle Befehle zur Cursorbewegung, zum Löschen und zum Einfügen. Für Suchen und Ersetzen gibt es eine äußerst leistungsfähige Wildcard-Verarbeitung. Bei unbeabsichtigtem Löschen einer Zeile hilft ein Undo-Befehl.

Die gesamte Datei wird auf Programm-Befehl auch gespeichert, geladen oder ausgedruckt; an einen Spooler wurde leider nicht gedacht. Auch die Ausgabe des Textes auf den Bildschirm ab beliebiger Position mit anderen Parametern stellt kein Problem dar.

Wichtige Editor-Parameter werden mit den »Set«-Befehlen eingestellt. Hier wird über den Refresh-Modus des Fensters und die Task-Priorität, aber auch über die Tastenbelegung entschieden. Außerdem kann ein EPL-Programm auch ein anderes aufrufen, komplexen Schachtelungen steht nichts im Wege. Entsprechend den normalen Tasten werden auch die Maustasten auf Wunsch mit Funktionen belegt. Eine eigene Scroll-Routine wie bei SuperEd oder CygnusEd ist somit kein Problem. Der normale Weg für die Maussteuerung ist allerdings die Programmierung von Menüs, die in allen Details

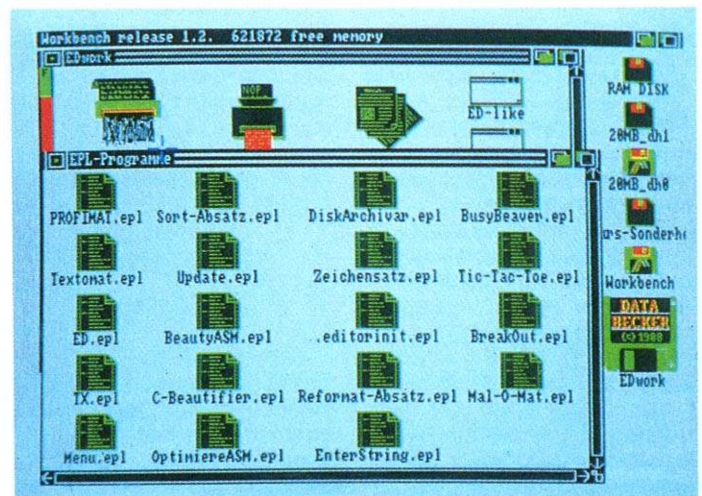


Bild 7. Bei EdWork werden viele Erweiterungsprogramme mitgeliefert

(Checkmarks, Disabling) unterstützt wird.

Außerdem ist man nicht auf ein Fenster beschränkt. Die Window-Befehle öffnen, verschieben, vergrößern, aktivieren und schließen beliebig neue Editor-Fenster. Und wem diese Befehlsvielfalt nicht reicht, der kann auch noch CLI-Befehle ausführen lassen.

Im Geschwindigkeitsvergleich ist EdWork schneller als ProEd, wozu allerdings nicht viel gehört. Die Ersetzen-Funktion zeigt den gleichen Nachteil (oder Vorteil, je nach Betrachtungsweise) wie CygnusEd: Der Text wird zur Kontrolle unmittelbar am Bildschirm ersetzt, was die Ausführungszeit in die Länge zieht, dafür aber dem Programmierer mehr Sicherheit bietet. EdWork ist hier aber immer noch dreimal schneller als CygnusEd.

Das Handbuch ist wohl das einzige in diesem Vergleich, das diesen Titel mit vollem Recht trägt. Es umfaßt 360 Seiten und ist im üblichen Data-Becker-Stil hergestellt. Es erklärt EdWork und seine Funktionen übersichtlich und von Grund auf, womit auch das Erlernen von EPL erleichtert wird.

Denn um in den Genuß eines wirklich guten Editors zu kommen, läßt sich die Programmierung nicht umgehen. Diese gibt Ihnen zwar ein zweifellos leistungsstarkes Werkzeug in die Hand, doch zahlreiche Besonderheiten des CygnusEd (zum Beispiel Autosave oder Hotstart) sind trotzdem nicht oder nur sehr schwer realisierbar. Bei CygnusEd müssen Sie zwar die Programmiersprache extra bezahlen, dafür wird aber der Umfang des Editor auch nicht damit belastet.

CygnusEd bietet eine unübertroffene Leistungsvielfalt mit Funktionen, die auch nach langer Arbeit immer noch positive Überraschungen zu bieten haben. Er ist für den engagierten Programmierer ein unverzichtbares Hilfsmittel. Mit EdWorks wurde eine sehr gute Idee verwirklicht, die Benutzeroberfläche im Eigenbau. Sie können sich jede beliebige Tastenbelegung definieren – besonders für den Umsteiger von anderen Computern eine nette Sache. Der SuperEd ist in seiner Preisklasse völlig konkurrenzlos. Vieles wurde verblüffend einfach verwirklicht, was entscheidend zur geringen Länge von 45 KByte beiträgt.

(Nikolaus Huber/so)

Anbieter:
ProEd: Atlantis, Postfach 1141, 5030 Hürth, Tel. 02233/41081
SuperEd: M&T-Verlag, Hans-Pinsel-Straße 2, 8013 Haar bei München
CygnusEd: Atlantis, Postfach 1141, 5030 Hürth, Tel. 02233/41081
ARexx (Programmiersprache zu CygnusEd): William S. Hawes, P. O. Box 308 Maynard, MA 01754, USA, Tel: 001/508 568 8695
EdWork: Data Becker GmbH, Merowingerstraße. 30, 4000 Düsseldorf



Lattice 5.0 kontert Aztec

Seit es den Amiga gibt, scheiden sich die Geister am »richtigen« C-Compiler. Lattice und Aztec beherrschen seit langem den Markt. Bisher sah es nach einem deutlichen Punktvorteil für Aztec aus. Kann Lattice mit seiner neuesten Version die Gunst der Programmierer gewinnen?

Sowohl der Lattice- (Bild 1) als auch der Aztec-Compiler (Bild 2) bieten in ihren neuesten Versionen einen enormen Leistungsumfang. Wir haben beide Compiler für Sie unter die Lupe genommen. Die Vor- und Nachteile der aktuellen Versionen schildert dieser Artikel. Außerdem geben wir Ihnen Tips, die Sie beachten sollten, um effektive Programme zu erhalten.

Wir wollen Ihnen hier nicht nur bei der Entscheidung für Ihren ersten C-Compiler helfen, sondern auch dem fortgeschrittenen C-Anhänger interessante Hintergrundinformationen geben, die für einen objektiven Vergleich der beiden Compiler unerläßlich sind – und trotzdem oft unbeachtet bleiben. Zum Test standen uns jeweils die neuesten Versionen zur Verfügung, der Aztec 3.6a und der Lattice 5.0, beide mit Source-Level-Debugger.

Unentbehrlich: Englischkenntnisse

Leider muß man nach dem Auspacken seines neuen C-Compilers (wie auch der meisten anderen Software-Pakete) immer noch feststellen, daß ohne fundierte Englisch-Kenntnisse »nichts geht«. Noch immer werden Handbücher in dieser Sprache verfaßt und selten genug für den deutschen Markt übersetzt. Aber wie ist es um die Qualität dieser Anleitungen bestellt?

Wir erhalten bei beiden Compiler-Paketen dicke Handbücher. Beide setzen voraus, daß man C bereits beherrscht. Natürlich kann ein Handbuch nicht dazu dienen, die Programmiersprache zu erlernen, beide geben jedoch entsprechende Literaturhinweise.

Bequemes Arbeiten

Lattice wartet mit einer etwas besseren Strukturierung des Handbuchs auf; es ist leichter zu lesen und erläutert die Installierung des Pakets besser. Auf mögliche Konsequenzen einiger Compileroptionen wird sofort hingewiesen. Bei Aztec hingegen erhält man entsprechende Informationen manchmal erst, wenn man die Bibliotheken durchstöbert hat.

Um mit einem Compiler »bequem« arbeiten zu können, sind gewisse Voraussetzungen nötig. Beispielsweise machen 512 KByte Speicher und nur ein Laufwerk die Arbeit oft genug zur Qual. Doch was ist zum Programmieren wirklich nötig?

Wir sollten bedenken, daß ein Editor, der Compiler, der Linker, die Include-Files und (beim Aztec) der Assembler ständig gebraucht werden. Außerdem muß natürlich der Speicherbedarf des zu bearbeitenden Programms mitberechnet werden. Müssen alle diese Programme jedesmal von Diskette geladen werden, dann macht sich dies in einer erheblichen Beeinträchtigung der Arbeitsgeschwindigkeit bemerkbar.

Um alles Benötigte im Speicher zu halten, ist ein enormer Speicherausbau nötig. Eine Festplatte ist eine weitere Möglichkeit, die Ladezeiten erheblich zu verringern. Der optimale Ausbau dürfte jedoch eine Kombination aus Festplatte und erweitertem Speicher sein.

Der Speicherbedarf des Aztec-Entwicklungssystems ist deutlich geringer als der von Lattice. Das zeigt sich schon daran, daß es möglich ist, alle nötigen Dateien auf eine einzige Diskette zu kopieren und damit zu arbeiten, ohne »Diskjockey« spielen zu müssen. Bei Lattice ist hier schon ein zweites Laufwerk zwingend erforderlich.

Als sehr hilfreich hat sich die Möglichkeit beider Compiler herausgestellt, Include-Files vorab zu compilieren, so daß oft gebrauchte Includes in einer einzigen Datei untergebracht werden. Das spart natürlich Speicherplatz auf Diskette und unnötige Zugriffe. Diese Funktion wurde bei Lattice erst seit der neuesten Version hinzugefügt. Sie ist nach unseren Erfahrungen für eine erträgliche Compilationszeit dringend notwendig, da der Zugriff auf viele Include-Files von Diskette manchmal eine mehrminütige Länge annehmen kann.

Man sollte sich vor Augen halten, daß man – zumindest mit dem Aztec – zwar mit 512 KByte Arbeitsspeicher und einem Laufwerk programmieren kann. Vernünftiges Arbeiten ist damit aber noch lange nicht gewährleistet. Wenn Sie alle ständig benötigten Programme im RAM halten und beim Compilieren und Linken nicht ständig auf Diskette zugreifen wollen, sollten Sie etwa 2,5 MByte und zwei Laufwerke besitzen. Allerdings werden Lattice-Anwender auch bei dieser Konfiguration den Wunsch nach einer Festplatte nur schwer unterdrücken können, wenn der Compiler voll genutzt werden soll.

16 oder 32 Bit?

Um Gurus zu vermeiden, hat es sich bei Aztec eingebürgert, jeden Compilerlauf mit der Option »+l« zu starten, so daß der Datentyp »int« immer mit 32 Bit Breite angelegt wird. Dies hat den unschätzbaren Vorteil, daß die meisten Public Domain-Quelltexte ohne Beanstandung zum Laufen gebracht werden. Außerdem erhöht man die Kompatibilität zum Lattice-Compiler, bei dem 32-Bit-Integer voreinstellt sind. Der Gebrauch dieser Option hat allerdings auch Nachteile. Diese werden beim Programmieren leicht übersehen, tragen aber allzuoft einen



Bild 1. Groß und mächtig – Lattice C

immensen Teil zu einem schlechten Compilat bei:
- Sämtliche Funktionsargumente werden als Langworte auf den Stack gebracht, egal welche tatsächliche Datenbreite vorhanden ist.
- Rückgabewerte von Funktionen sind grundsätzlich Langworte.
- Die Berechnung eines Ausdrucks erfolgt ausschließlich auf der Basis von 32 Bit, selbst wenn »short int« oder »char« angegeben ist.
- Konstanten werden immer als Langwort angesehen.

Das bringt so manche unnötige Umrechnung mit sich, da nicht als Langworte deklarierte Variablen stillschweigend mit dem Maschinenbefehl »EXT« vorzeichenrichtig auf 32 Bit erweitert werden müssen. Beispielsweise müßte ein 16-Bit-Funktionsargument erst aus seiner Speicherzelle in ein Datenregister geladen, dann auf 32 Bit erweitert und schließlich auf dem Stack abgelegt werden. Das sind zwei zusätzliche Anweisungen, die Zeit und Speicherplatz kosten. Noch schlimmer wird es, wenn man mit 8 Bit, also mit dem Typ »char«, arbeitet, bei dem dann schon zwei »EXT«-Anweisungen notwendig sind. Wie man sieht, bekommt man mit kleinerer Datenbreite keinen schnelleren Code, eher ist das Gegenteil der Fall. Es ist also sinnvoll, 32-Bit-Variablen zu benutzen, soweit dies möglich ist. Dieser Sachverhalt gilt natürlich für beide Compiler, wie Bild 3 und 4 verdeutlichen.

Eine andere Möglichkeit ist es, bei Aztec die »+l«-Option wegzulassen oder bei Lattice »-w« zu ergänzen, so daß intern 16 Bit angenommen werden und unnötige Langwortoperationen wegfallen. Der Programmcode wird so kompakter, gern compilieren. Der Lattice-Compiler kann hier übrigens sehr gut Hilfestellung geben, wie wir gleich noch sehen werden.

Allen Programmierern verhaßt sind die Compilerwarnings. Das sind Hinweise des Compilers auf Unsauberheiten im Code, die eventuell zu Problemen während der Laufzeit führen können. Dies ist auf dem Amiga in der Tat ein Problem, das nicht so leicht in den Griff zu bekommen ist. Man könnte die Warnings zwar einfach mit einer Compileroption ausblenden, aber ob dies der richtige Weg ist? Hinsichtlich eines sauberen Programmierstils läßt sich diese Frage sicher verneinen. Aber selbst diesbezüglich ehrgeizigere Freaks sollen schon gescheitert sein, um dann doch reumütig drei Zeichen mehr ins CLI zu tippen und die Warnings doch noch abzuschalten.

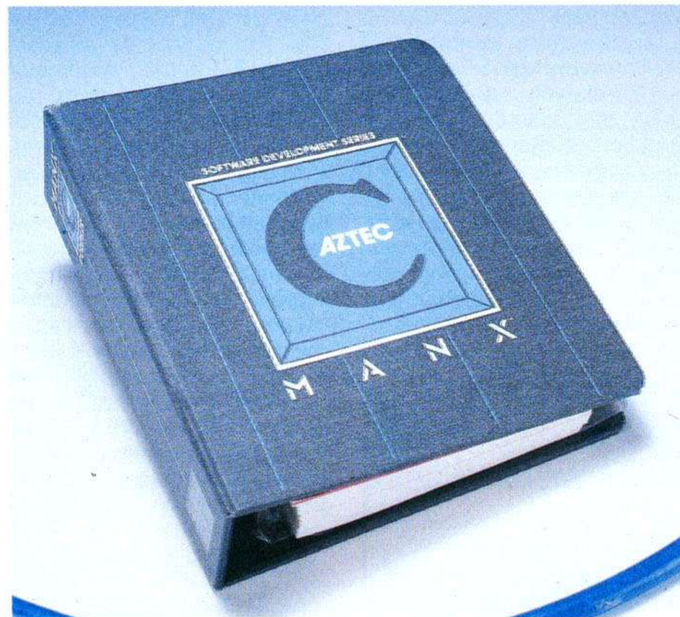


Bild 2. Kompakte Kraft – Aztec C

bei tiefer Rekursion wird der Stack im günstigsten Fall nur um die Hälfte belastet. Das klingt zwar schön, erfordert aber auch eine sehr sorgfältige Programmierung – besonders wenn es um den Aufruf von Amiga-spezifischen Funktionen geht. Man muß jetzt nämlich im Hinterkopf behalten, daß der Compiler Funktionsargumente genau ihrem Typ entsprechend stapelt, eventuelle Konstanten werden als Wort behandelt. Aber alle Amiga-Funktionen erwarten 32 Bit, auch wenn man selbst nur 16 Bit vermutet. Das gleiche gilt für die Funktionswerte, die nun, wenn nicht ausdrücklich anders angeordnet, als Wort zurückgegeben werden. So kann es bei Unachtsamkeit schon mal passieren, daß ein Pointer bei der Rückgabe halbiert wird (was den Guru weckt). Deshalb lassen sich so viele Public Domain Programme nur mit 32-Bit-Integer compilieren.

Der Aztec-Code

Dieses C-Listing...

```
short a,b,c;
main()
{     a=a*(b+c);
      func(a);
}
```

...übersetzt Aztec mit der »+l«-Option zu folgendem Assemblerlisting:

```
move.w  _b,d0
ext.l   d0
move.w  _c,d1
ext.l   d1
add.l   d1,d0
move.w  _a,d1
ext.l   d1
jsr             .mulu#
move.w  d0,_a
move.w  _a,d0
ext.l   d0
move.l  d0,-(sp)
jsr             _func
add.w   #4,sp
```

Einen wesentlich effektiveren Code erhält man aber mit 16-Bit-Integer, wenn man diese Option wegläßt:

```
movem.l .3,-(sp)
move.w  _b,d0
add.w   _c,d0
muls.w  _a,d0
move.w  d0,_a
move.w  _a,-(sp)
jsr             _func
add.w   #2,sp
```

Bild 3. Aztecs »+l«-Option birgt auch Nachteile in sich...

Compilerwarnings einfach ignorieren?

Dabei ist das Problem immer das gleiche: Es werden Werte an Zeiger übergeben, die nicht vom gleichen Typ sind, zumindest aus der Sicht des Compilers. Und genau dies wird dem Programmierer mitgeteilt. Bei der Fülle von verschiedenen Strukturen und deren Pointern im Amiga-Betriebssystem kann man leicht den Überblick verlieren. Grundsätzlich läßt sich dieses Problem aber umgehen, indem man bei den entsprechenden Anweisungen eine explizite Typumwandlung mit Hilfe des Cast-Operators durchführt. Allerdings kann der Programmcode dadurch unübersichtlicher werden. Bei Aztec bietet sich daher folgende Methode an: In dem Ordner für die Includes befindet sich ein File namens »functions.h«, das sozusagen den Compiler über die Rückgabetypen sämtlicher Amigafunktionen aufklärt. So wird schon ein sehr großer Teil potentieller Warnings vermieden.

Prototypes und Functions

Wenn man sich für die 16-Bit-Integer-Verarbeitung entschieden hat, ist eine korrekte Funktionswertrückgabe garantiert. Beim Lattice-Compiler bindet man entsprechend der benutzten Funktionen Include-Files aus dem Ordner »proto« ein. Diese übernehmen unter anderem das sogenannte »Argument Type Checking«, auch »Functions Prototyping« genannt, also die Überprüfung der Funktionsargumente auf ihren richtigen Typ. Damit werden bei der Programmierung viele Fehler schon im voraus aufgedeckt.

Der Lattice-Code

Auch bei Lattice ließe sich dieses

```
MOVE.W  01.00000002(A4),
        D0
EXT.L   D0
MOVE.W  01.00000004(A4),
        D1
EXT.L   D1
ADD.L   D1,D0
MOVE.W  01.00000000(A4),
        D1
EXT.L   D1
JSR         __CXM33(PC)
MOVE.W  D0,01.00000000
        (A4)
EXT.L   D0
MOVE.L  D0,-(A7)
JSR         _func(PC)
ADDQ.W  #4,A7
```

durch einen besseren Code ersetzen:

```
MOVE.W  01.00000002(A4),
        D0
ADD.W   01.00000004(A4),
        D0
MOVE.W  01.00000000(A4),
        D1
MULS.W  D0,D1
MOVE.W  D1,01.00000000
        (A4)
MOVE.W  D1,-(A7)
JSR         _func(PC)
ADDQ.W  #2,A7
```

Bild 4. ...während Lattices »-w« nur Vorteile mit sich bringt

Zu guter Letzt gibt es noch Funktionen, bei denen verschiedene Typen zurückgegeben werden können. Beispielsweise könnte OpenLibrary() einen Pointer auf IntuitionBase, auf GfxBase, auf DosBase etc. liefern, so daß tatsächlich nur noch der Cast-Operator einer Warnung zu entrinnen hilft.

Es bleibt festzuhalten, daß bei Lattice ein lausiger Programmierstil schwieriger aufrechtzuerhalten ist als bei Aztec. Es wird ganz klar der ANSI-Standard unterstützt, Fehlermeldungen bei Lattice können bei Aztec eventuell nur als Warnung quittiert werden. Das ist für den C-Anfänger vorteilhaft, wenn es darum geht, die Sprache wirklich nach den Richtlinien von Kernighan & Richie zu lernen und hilft letztendlich, Fehler zu vermeiden. Äußerst positiv macht sich bei Lattice auch bemerkbar, daß man hinsichtlich eines optimalen Codes auf nicht benutzte Variablen oder auf nicht initialisierte Autovariablen (das sind lokale oder Register-Variablen, die nicht automatisch initialisiert werden, sondern zufällige Werte beinhalten können) hingewiesen wird.

Geschwindigkeit ist Trumpf

Für die meisten Programmierer am interessantesten ist die Frage nach der Ablaufgeschwindigkeit eines fertig compilierten Programmes. Objektive Benchmarks zu finden ist immer problematisch. Wir haben dazu ein bekanntes Benchmark-Programm namens »DHRYSTONE« gewählt. Es testet beide Compiler in verschiedener Hinsicht und vermittelt einen weitgehend objektiven Eindruck. Es werden zu 53 Prozent Zuweisungen, zu 32 Prozent Kontrollanweisungen und zu 15 Prozent Funktionsaufrufe untersucht (siehe Tabelle 1). Viele »dhry/sec« bedeuten dabei ein besseres Compilationsergebnis.

Zwar bietet Lattice 5.0 Funktionen an, mit deren Hilfe sich der Code noch weiter optimieren ließe, aber ein Geschwindigkeitsvorteil gegenüber Aztec ist auch ohne diese klar ersichtlich. Hierzu zählen der »Global Optimizer« der beim Lattice überflüssige Anweisungen eliminiert. Normalerweise führt Lattice auch eine Stack-Überlaufkontrolle durch, die das Programm etwas verlangsamt, jedoch mit der »-v«-Option abgeschaltet werden kann. Das erspart je nach Anzahl von benutzten Funktionen auch eine Menge Speicherplatz. Diese Option sollte allerdings erst benutzt werden, wenn das Programm ausgiebig auf Fehler überprüft wurde. Bei Aztec finden keine derartigen Überprüfungen während der Laufzeit statt.

Üblicherweise werden in C die Parameter eines Funktionsaufrufs einfach auf den Stack gelegt und dann die passende Funktion aufgerufen. Sämtliche Amiga-spezifische Funktionen verlangen die Argumente aber in Registern, so daß eine große Library mit entsprechenden Interface-Routinen notwendig ist, die die Verbindung zwischen Parameterübergabe auf dem Stack und den Registern übernimmt. Das wirkt sich natürlich negativ auf die Laufzeit des Programmes aus, da zusätzliches Umladen von Speicher- und Registerinhalten notwendig ist.

Bei Lattice wird dies vom Compiler umgangen, indem die Argumente direkt in die Register geladen werden. Der fertige Programmcode erweckt fast den Eindruck, in Assembler geschrieben zu sein und es kann beim Linken sogar auf die »Amiga.lib« verzichtet werden, wobei dann nur noch wenige Sekunden Linkzeit in Anspruch genommen werden! Vor allem Anhänger von rechenintensiven Anwendungen wie Computergrafik werden bei solchen Geschwindigkeitsvorteilen doch eher mit Lattice liebäugeln (siehe Bild 5). Lediglich bei einer sehr langen Parameterliste muß noch der Umweg über den Stack genommen werden, da selbst der Amiga nur 16 Register besitzt.

DHRYSTONE-Benchmark 1.1

	Optionen	Codelänge	Recheneinheiten	Erklärung
Aztec 3.6a	+L	7416 Byte	632 dhry/sec	32-Bit-Integer
Lattice 5.0	keine	8748 Byte	746 dhry/sec	32-Bit-Integer
Lattice 5.0	-O	8612 Byte	941 dhry/sec	mit Global Optimizer
Lattice 5.0	-O -v	8240 Byte	1041 dhry/sec	ohne Stack-checking

Tabelle 1. Lattice C macht als Sprinter die bessere Figur

Heiko Schlichting



sammelte seine ersten Computererfahrungen 1984 mit einem PET 2001 und später mit einem C 64. Bereits nach relativ kurzer Zeit wuchs der Wunsch nach einem Computer, der bessere Grafikfähigkeiten mitbringt und in Hochsprachen programmiert werden kann. Der Amiga 2000 war für beides prädestiniert.

Heiko Schlichting studiert Biochemie an der Freien Universität Berlin.

Er sammelte intensive Erfahrungen mit der Sprache C auf einer Unix-Anlage, dem PC und mit Hilfe des Lattice-C auf dem Amiga. Dessen neueste Version testete er für diesen Artikel.

»Make« macht's bequemer

Wer mit längeren Programmierprojekten beschäftigt ist, der wird die Hilfe des von Unix abgekupferten Make-Utilities zu schätzen wissen. Mat hat so die Möglichkeit, mittels eines einzigen Befehls, – »make« bei Aztec oder »lmk« bei Lattice – das Compilieren, Assemblieren, Linken und eventuelle Aufräumarbeiten (»CleanUp«) automatisch erledigen zu lassen. Dazu stehen in einem selbst geschriebenen »makefile« die genauen Instruktionen, die von »make« ähnlich einem Batch-File interpretiert werden. Ein besonderer Vorteil besteht darin, daß ein aus mehreren Modulen bestehendes Programm nicht vollends neu compiliert werden muß, wenn nur ein Teil des Quelltextes geändert wurde. »Make« realisiert dies, indem die letzte Zugriffszeit der Objekt-Files mit der des C-Quelltextes verglichen wird. Ist das C-File jüngeren Datums, dann ist das Objektmodul nicht mehr »up to date« und wird neu compiliert. Zur korrekten Arbeitsweise muß natürlich immer die aktuelle Uhrzeit vorhanden sein.

Weiterhin wird so die Tastatur geschont, denn »make« erspart das wiederholte Eintippen von langen Compileraufrufen. Es scheint in der PD-Scene offensichtlich schon üblich zu sein, zu jedem Source-Code ein entsprechendes »makefile« zu liefern – zur Freude eines jeden C-Programmierers, der sich dann nicht weiter um die Compileroptionen kümmern muß. Das »make«-Utility ist auf jeden Fall bei beiden Compilern eine sehr empfehlenswerte Hilfe.

Der Editor

Ein Editor ist zum Schreiben eines C-Quellcodes unerläßlich. Beide Compiler haben einen Editor bereits im Lieferumfang enthalten. Diese unterscheiden sich allerdings stark in der Leistungsfähigkeit. Der Editor, der bei Aztec mitgliefert wird (genannt »Z«), ist dem Unix-Editor »vi« sehr ähnlich. Die Bedienung des Editors ist demnach auch in keiner Weise auf das Betriebssystem des Amiga zugeschnitten. Das macht die Bedienung kompliziert und erscheint unnötig umständlich. Die gesamte Leistung des »Z« übersteigt die des »ED« nicht deutlich, wenn man auch eine sehr schöne Möglichkeit geschaffen hat, Funktionen zu suchen. Wenn mit dem mitgeliefertem Utility (»CTags«) eine Referenzliste erstellt wurde, kann man mit einem Befehl vom Funktionsaufruf zur Funktionsdefinition springen – sogar wenn die Funktion in einem anderen Textfile definiert ist.

Der Lattice-Editor »LSE« wurde für die Version 5.0 deutlich verbessert und ist leichter zu bedienen als der »Z«. Viele Operationen sind über Menüs und Funktionstasten aufrufbar, für Mausanhänger und Tippfaule ist das sicher von Vorteil. So können Compiler und Linker aus dem Editor mit einem Tastendruck gestartet werden, eventuelle Fehler werden dann im Editor angezeigt, wobei sich der Cursor schon an der Stelle befindet, an der der Fehler auftritt.

Einen großen Nachteil haben jedoch beide Editoren: Sie erlauben es nicht, Umlaute in den Quelltext einzufügen. Zwar kommen in der Syntax der Sprache C keine Umlaute vor, doch bei der Textausgabe und bei Kommentaren verzichtet man nur ungern darauf. Die

wohl ursprünglich als Eingabehilfe gedachte Beschränkung wird hier zur Qual.

Viele Programmierer wollen sich für verschiedene Aufgaben nicht jedesmal an einen neuen Editor gewöhnen. Sie verwenden nur einen, der an verschiedene Gegebenheiten angepaßt werden kann. Auf Seite 8ff. in diesem Heft stellen wir einige leistungsfähige Editoren vor. Wer sich die Ausgaben für einen teuren professionellen Editor sparen möchte, findet auf der Fish-Disk 153 einen für C-Programmierer sehr gut geeigneten Editor namens »DME«. Dieser erlaubt es sogar, seine Funktionsvielfalt durch eigene Befehle zu ergänzen. Die Belegung der Tasten kann komplett geändert werden und auch Menüs können nach eigenen Wünschen aufgebaut oder ganz weggelassen werden. Viele Befehle des »DME« sind ganz besonders auf die Wünsche von Programmierern zugeschnitten; so ist es zum Beispiel möglich, eine Referenz zu Begriffen anzeigen zu lassen. Steht der Cursor auf dem Wort »Screen«, so läßt sich mit einem Tastendruck ein weiteres Window erzeugen, in dem die Screen-Struktur mit allen Flags angezeigt wird. Der »DME« unterstützt außerdem auch die oben

Es geht auch in Assembler

erwähnte »CTags«-Funktion des Aztec-Compilers. Ein Aufruf von Compiler, Linker und »make«-Utility direkt aus dem Editor ist durch die hohe Flexibilität ebenfalls möglich. Dieser Public Domain-Editor eignet sich viel besser zur Programmeingabe als die beiden bei den Compilern mitgelieferten. Besonders Aztec-Benutzer werden sich schnell nach einem leistungsfähigeren Editor umsehen.

Beide Entwicklungssysteme stellen einen komfortablen Makroassembler zur Verfügung, wobei nur der Aztec-Assembler grundsätzlich zur Erzeugung eines Objektfiles benötigt wird. Dies liegt daran, daß bei Aztec der C-Source mittels Compiler zunächst einmal in ein Assemblerlisting übersetzt wird. Lattice erzeugt dagegen einen eigenen Zwischencode, der von Pass 2 dann auf das Objektformat gebracht wird. Dem Aztec-User bleibt so natürlich die Möglichkeit offen – sofern entsprechende Assemblerkenntnisse vorhanden sind – nachträglich Optimierungsarbeiten durchzuführen, zumal man sich das Assembler-Listing mit der Option »-T« auch noch übersichtlich gestalten lassen kann. Hierbei wird der C-Sourcecode als Kommentar an die passende Stelle im Assembler-Listing eingefügt. Praktisch ist auch die Möglichkeit beim Aztec, zwischen den Präprozessoranweisungen »#asm« und »#endasm« Assembleranweisungen sogar innerhalb des C-Quelltextes zuzulassen.

Beide Assembler können als in sich abgeschlossen angesehen werden, sie sind also auch bestens zur ausschließlichen Assemblerprogrammierung geeignet. Vor allem können die von Commodore gestellten Include-Files gelesen werden, die Teil des Lieferumfangs sind. In der Version 5.0 enthält das Lattice-Handbuch nun auch die nötigen Informationen zum Assembler.

Lattice verwendet Register

Der Funktionsaufruf

```
Text(RastPort,text,len);
```

erfolgt bei Lattice per Registerübergabe:

```
MOVEA.L 01.00000000(A4),
        A1      ;Rast-
                 Port-
                 Zeiger
                 in a1
OVEA.L 01.00000008(A4),
       A0      ;Zeiger
       auf Text in a0
MOVE.L  01.00000004(A4),
        D0      ;Länge
                 in d0
MOVEA.L _GfxBase(A4),
        A6 ;GfxBase in
            a6
JSR        FFC4(A6)
      ;und auf geht's...
```

Bild 5. Selbst in Assembler geht's nicht besser

Mehr Effektivität mit weniger Speicher?

Im Zusammenhang mit den beiden Assemblern möchten wir näher auf die sogenannten Speichermodelle eingehen, die bei Aztec auch als »Small Code« oder »Large Code« beziehungsweise »Small Data« oder »Large Data« bezeichnet werden, was vielleicht nur für diejenigen von Bedeutung sein mag, die neben C auch in Assembler programmieren. Aber auch für C-Freaks ist dies durchaus interessant. Normalerweise können Variablen im Speicher über ihre volle Langwortadresse angesprochen werden, wozu der Prozessor bei einem Zugriff erst einmal das entsprechende Langwort aus dem Speicher holen muß (»Large Data«). Eine entsprechende Assembleranweisung könnte etwa so aussehen:

»move.l $20bd50,d0«

Dabei wird der Wert der Variablen in das Datenregister »d0« geladen. Um nun ein sowohl schnelleres als auch kompakteres Programm zu erhalten, hat man sich für eine andere Methode entschieden: Sämtliche Programmdaten liegen in einem einzigen Speicherblock, der nicht größer als 64 KByte sein darf. So wird nur ein Zeiger auf diesen Block benötigt, bei einem Zugriff auf Daten ist nur noch eine 16-Bit-Referenz nötig (16 Bit entspricht dabei dem Zugriffsraum von 64 KByte, deswegen auch »Small Data«). Zwangsläufig muß der Zeiger dann ständig parat sein, der bei Aztec und Lattice automatisch in Adreßregister »a4« gelegt wird. Ein typischer Variablenzugriff sieht so aus:

»`move.l ffe2(a4),d0`«

Dieser Befehl erzielt die gleiche Wirkung wie der oben genannte. Beachten Sie bitte, daß der Zugriff auf Auto-Variablen in gleicher Weise realisiert wird, es wird lediglich Register »a5« (statt »a4«) in Beschlag genommen.

Ähnlich sieht es bei Aufrufen von Funktionen aus. Anstatt diese beispielsweise mit »jsr $202f60«, also »Large Code«, über ihre volle Adresse anzuspringen, benutzt man die PC-relative Adressierungsart, so daß bei »jsr 2afe(pc)« nur eine 16-Bit-Referenz nötig ist. Dies wird allerdings uneffektiv bei Programmen, in denen die 16 Bit nicht mehr ausreichen, da hier sogenannte Sprungtabellen angelegt werden und mehrere Sprünge nacheinander durchgeführt werden müßten. »Small Code« und »Small Data« sind nun auch bei Lattice Voreinstellung, die erzeugte Codelänge wird dadurch schon sehr an die des Aztec angeglichen.

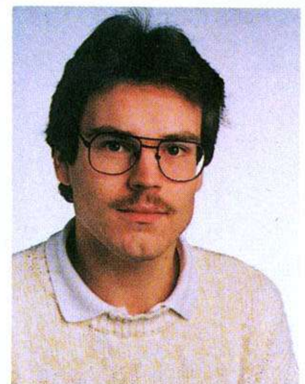
Das »Small-Data«-Modell macht Probleme , wenn es darum geht, fremde Assemblerlistings zu verwenden, wie zum Beispiel die des Seka-Assembler. Dieser unterstützt »Small Data« nicht automatisch, und dessen Programmierer haben demnach das Register »a4« zur freien Verfügung. Das hätte natürlich unweigerlich Konsequenzen, die aber sowohl bei Lattice als auch Aztec mit der Benutzung von »Large Data« verschwinden würden.

Erwähnenswert bei Aztec ist die Möglichkeit, die verschiedenen Speichermodelle auch innerhalb des Assemblerlistings wechseln zu können; es werden hier entsprechende Direktiven angeboten.

Verbesserter BLink

Inzwischen ist auch der Linker von Lattice soweit verbessert worden, daß das Aztec-Pendant teilweise sogar übertroffen wird. So geht Blink in seinen Funktionen, die die Veränderungen der Hunks des Output-Files beeinflussen, sogar so weit, daß die Größe des Code-Hunks einen selbst zu bestimmenden Wert nicht überschreiten darf.

Christian Wolf



ist Amiga-Freak der ersten Stunde. Der Amiga 1000 löste einen Atari 800XL ab, der zum Programmieren zu klein geworden war. Seit Beginn seiner Amiga-Karriere stützt er sich auf das Aztec-Entwicklungssystem.

Christian Wolf studiert Informatik an der Technischen Universität in Berlin. Er testete den Aztec-C-Compiler für diesen Artikel

Neben mehreren Programmen im 68000er-Magazin und den Amiga-Sonderheften hat er die Software für den »AL-COMP«-Soundsampler entwickelt. Er hat dabei den Aztec-C-Compiler bis in die letzten Feinheiten kennengelernt.

Als Folge dessen kann der Programmcode in mehreren Hunks untergebracht werden. Dies ist zum Beispiel sinnvoll, wenn das fertige Programm auch unter extremer Speicherknappheit laufen soll, da der Programmcode dann auf kleine Speicherstücke verteilt werden kann. Aztec läßt hier leider nur

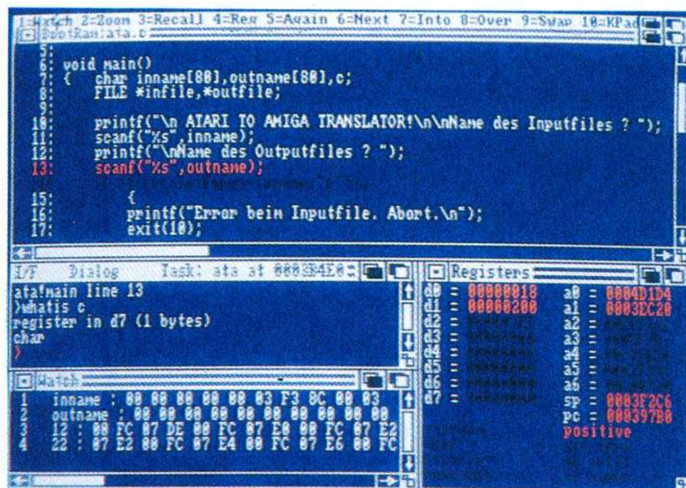


Bild 6. Reich an Funktionen präsentiert sich der Lattice Debugger

eine konstante Größe von 8 KByte zu. Allerdings ist der Einfluß auf dieses sogenannte »Amiga-Scatter-Load-Format« nur für wenige Programmierer von Interesse.

Großen Fortschritt für den C-Programmierer stellen die Source-Level-Debugger dar. Beide Compiler bieten ein solches Utility-Programm zu ihrer neuesten Version an.

Ein Source-Level-Debugger ist ein Hilfsmittel, mit dem man den Ablauf eines C-Programms Anweisung für Anweisung verfolgen kann. Dabei können auch Variablen per Namen ausgegeben oder verändert werden. Um bei langen Programmen bestimmte Teile untersuchen zu lassen, kann man sogenannte »Breakpoints« setzen, an denen der Debugger in den Einzelschrittmodus umschaltet. Wer mit der C-Syntax schon vertraut ist, wird sich freuen, daß die Debugger bei der Ausgabe der Variablen und bei deren Manipulationen darauf zurückgreifen. Veränderungen lassen sich während des Debuggings vornehmen, wie man es vom Programmieren gewohnt ist. Ein komfortabler Einblick in die Strukturen des Betriebssystems ist während des Programmablaufs dadurch möglich, daß Strukturen in ihren Komponenten per Namen angezeigt werden können.

In ihrer Funktionsvielfalt unterscheiden sich die Source-Level-Debugger von Aztec und Lattice nur wenig, beide unterstützen die Umschaltung von C-Sourcetext auf Assembler. Der Source-Level-Debugger von Lattice ist jedoch für den Anfänger leichter zu bedienen, da viele Befehle über Menüs und Funktionstasten ausgeführt werden können und nicht immer die Kommandos über Tastatur eingegeben werden müssen. Auch das Arbeiten mit mehreren Fenstern auf einem eigenen Screen (Bild 6) hat sich als praktisch und einfach herausgestellt. Bei Aztec hat man zum Teil Schwierigkeiten mit den Fenstern (Bild 7), da in dem Sourcetext-Fenster nicht horizontal gescrollt werden kann, so daß formatierte Texte unter Umständen nicht zu lesen sind. Der Speicherbedarf der beiden Programme ist so groß, daß sinnvolles Arbeiten erst ab einem Megabyte Speicher möglich ist. Für die Fehlersuche in C-Programmen sind diese beiden Utilities so hilfreich, daß man sie nach kurzer Zeit schon nicht mehr vermissen möchte. Wir können die beiden Source-Level-Debugger zur schnellen und komfortablen Fehlersuche aus eigener Erfahrung nur empfehlen.

Nur bei Lattice

Wir möchten hier noch auf einige Besonderheiten des Lattice 5.0 eingehen, die bei Aztec nicht zu finden sind, wie zum Beispiel die sehr ausführlich dokumentierten Include-Files. Man findet zu fast allen Strukturen des Amiga-Betriebssystems eine Erläuterung der Komponenten und der Flags, so daß nicht so oft zum Handbuch gegriffen werden muß. Hinsichtlich der Benutzung des oben erwähnten Reference-Befehls im »DME« ist dies eine noch größere Erleichterung.

Der Lattice Compiler verwendet automatisch Prozessorregister für Auto-Variablen, ohne daß dies explizit mit »register« angegeben werden muß. Es wurden zusätzliche Schlüsselworte eingeführt, die den Bedürfnissen der Amiga-Programmierer Rechnung tragen. So ist es zum Beispiel möglich, ausgewählte globale Variablen schon im Quelltext mit dem Keywort »chip« zu versehen und festzulegen, daß diese dann in den gleichnamigen Speicher gelegt werden. Dies ermöglicht (im Gegensatz zum Compilieren mit der »+c«-Option), die restlichen Variablen im Fastmemory zu belassen.

Die Vielzahl von Library-Funktionen bei Lattice lassen beim Programmierer fast keine Wünsche mehr offen. Es finden sich viele von Unix, Xenix und natürlich vom ANSI-Standard bekannte Routinen. Es ist auch erfreulich, daß in den Libraries Funktionen implementiert wurden, welche die neue Workbench 1.3 nutzen, so daß unter anderem die Erstellung residenter Module oder die Verwendung von Environment-Variablen ermöglicht wird. Die Ausstattung des Aztec wirkt auf diesem Gebiet vergleichsweise spartanisch.

Dementsprechend sind natürlich auch Compiler und Linker »resident«-fähig, was Besitzer einer Speicherweiterung mit Aussicht auf schnellere Compilationszeiten sicher zu schätzen wissen.

Der Lattice-Compiler hat in vielen Aspekten die Nase vorn. Dies heißt aber nicht, daß bisherige Aztec-Freaks nun ihre gewohnte Programmierumgebung aufgeben sollen. Mit dieser war für Besitzer einer 2-MByte-Erweiterung bisher schon ein schnelles und zuverlässiges Arbeiten möglich. Was das Handbuch versprach, wurde vom Compiler auch prompt gehalten. Für eine ähnlich komfortable Umgebung könnte bei Lattice nur eine Festplatte sorgen, die die Ladezeiten verkürzt. Des weiteren ist das Gespann Aztec-C und Assembler für denjenigen zu empfehlen, bei dem Geschwindigkeitsprobleme ohnehin in Maschinensprache gelöst werden, da ein optimales Zusammenspiel garantiert ist. Wer dies erst einmal kennengelernt hat, möchte es so schnell nicht wieder missen. Auch ist der Aztec für den Gelegenheitsanwender mit geringem Speicher und einem Laufwerk die einzige Möglichkeit, in C zu programmieren.

Lattice verfolgt ganz klar eine sehr professionelle Linie, und mit der neuesten Version 5.0 werden dem Programmierer viele nützliche Hilfsmittel zur Verfügung gestellt. Es ist allerdings eine sehr genaue Kenntnis der Compileroptionen nötig, da sonst sehr leicht Abstürze erzeugt werden können. Nach unseren Erfahrungen ist eine gewisse Einarbeitungszeit nötig, bis man wirklich das gewünschte Resultat erhält. Danach erwies sich der Compiler jedoch als sehr zuverlässig, alle bisher genannten Features konnten tatsächlich nachvollzogen werden. Es gibt zwar Anzeichen dafür, daß noch ein paar kleine Bugs vorhanden sind, was jedoch keinen davon abhalten sollte, sich für diesen faszinierenden Compiler zu entscheiden.

Es gibt genügend Argumente sowohl für den einen wie auch für den anderen Compiler. Für welchen Sie sich entscheiden, ist zu einem großen Teil Geschmackssache. Denn auch wenn Lattice aus diesem Duell vorerst als Punktsieger hervorgeht, ist das Rennen noch lange nicht gelaufen. Man darf gespannt sein auf den nächsten Lauf. *(Christian Wolf/ Heiko Schlichting/so)*

Lattice-C 5.0; Preis: ca. 600 Mark; Hersteller: Lattice Incorporated, 2500 S. Highland Avenue, Lombard IL 60148, USA
Aztec-C 3.6a; Preis: ca. 600 Mark; Hersteller: Manx Software Systems Inc., P.O.Box 55, Shrewsbury, NJ 07701
Beide C-Compiler erhalten Sie beim Softwareversand Philgerma, Barerstr. 32, 8000 München 2, Tel. 089/281228 oder Baroper Str. 337, 4600 Dortmund, Tel. 0231/759292.

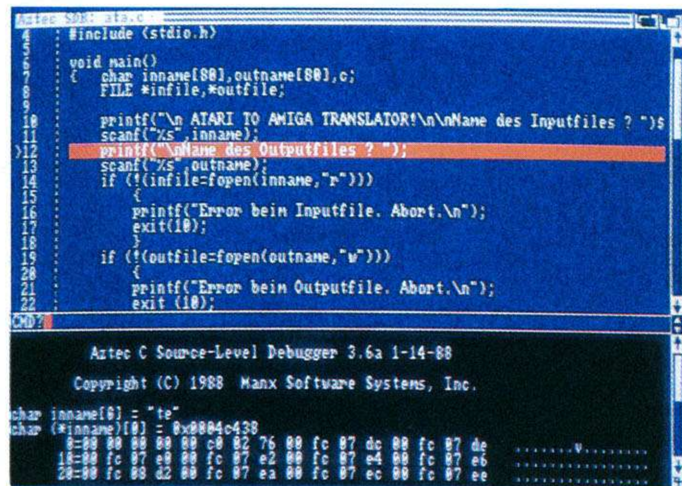


Bild 7. Aztecs Source-Level-Debugger in Action



228/903



Quo vadis, Lattice?

Lattice beschreitet mit »C++« völlig neue Wege. Diese Programmiersprache wird unter Insidern bereits als Nachfolger für »C« gehandelt. Wir verraten Ihnen, ob das wirklich schon der Anfang vom Ende der C-Ära ist.

Seit einiger Zeit ist die objekt-orientierte Programmierung ihren »LISPelnden« Kinderschuhen entwachsen. Immer mehr moderne Programmiersprachen weisen entsprechende Fähigkeiten auf. Was anfangs nach einem Spielzeug für die Pioniere der Künstlichen Intelligenz aussah, hat sich inzwischen zu einem ernstzunehmenden Faktor herauskristallisiert. Lattice hat die Zeichen der Zeit erkannt und kämpft mit »C++« (Bild 1) stellvertretend für die »sCene« um den Anschluß. Doch worum handelt es sich bei dieser neuen Sprache?

C++ wurde in den Bell Laboratories von AT&T in Los Angeles entwickelt. Dabei stand der Wunsch Pate, Schwächen von C auszumerzen und die Sprache um die Möglichkeiten der objektorientierten Programmierung zu erweitern. Die volle Kompatibilität zu C sollte unbedingt gewahrt bleiben. Denn mit der immer größer werdenden Popularität von C ist eine große Anzahl von Bibliotheken entstanden, von denen ein großer Teil als nahezu perfekt angesehen werden kann. Als Umsteiger kann man sich daher sofort auf die neuen Eigenheiten konzentrieren, so daß dieser Sprache praktisch die besten Voraussetzungen in die Wiege gelegt wurden.

C++ erinnert in vielem an die Hochsprachen Modula oder Pascal. Wer diese Sprachen kennt, wird folgende Punkte sehr zu schätzen wissen:

- Es ist wie in Pascal möglich, mit der »const«-Anweisung Konstanten zu definieren. Die »#define« Anweisung ist beinahe überflüssig. Sie ist allerdings aus Kompatibilitätsgründen weiterhin enthalten.
- Es gibt den neuen »::«-Operator, mit dessen Hilfe man auf gleichnamige Variablen auch innerhalb von geschachtelten Blöcken zugreifen kann. Auf diese Weise könnte eine globale Variable angesprochen werden, die in der gerade ausgeführten Funktion aber denselben Namen existiert.
- Das Funktions-Prototyping ist Standard geworden und erlaubt die genaue Überprüfung der Funktionsargumente.
- Es lassen sich sogenannte Adreß-Variablen vereinbaren, die, nachdem ihnen eine Variable zugewiesen wurde, bei Zugriff diese verändern. Damit ist es bei Anwendungen nicht mehr notwendig, mit Zeigern auf Objekte zu arbeiten, obwohl dies intern nach wie vor so gehandhabt wird.

In C++ wurden weitergehende Möglichkeiten geschaffen, komplexe Datentypen zu definieren. So kann sich der Programmierer sogenannte »Classes« aufbauen, die wie »Structures« aus verschiedenen elementaren Komponenten bestehen. Hier wird aber peinlich genau zwischen dem »Private«- und dem »Public«-Teil der »Class« unterschieden, so daß gewisse Komponenten nur für interne Zwecke zugänglich sind. Dies ist hilfreich, weil sich in der Class auch Funktionen angeben lassen, die auf diese Komponenten angewandt werden können.

C++ als Pre-Compiler

Des weiteren ist es erlaubt, in den Classes den Operatoren (beispielsweise auch »+« oder »–«) eine neue Funktion zuzuordnen, so daß zu den geschaffenen Objekten eine eigene Arithmetik aufgebaut werden kann. Sie können sicher erahnen, welch faszinierende Möglichkeiten diese Sprache bietet.

Wie wir schon gesehen haben, ist C++ nichts anderes als eine Erweiterung für die Sprache C. Konsequenterweise liefert Lattice seinen C-Compiler in der Version 4.01 mit Blink 7.0 und einem C++ Pre-Compiler, der den »C++«-Source in ein C-Programm übersetzt. Wer den Lattice C-Compiler besitzt, der erhält C++ auch günstiger als Zusatzpaket. Dadurch kann auch die neue Version 5.0 benutzt werden. Zum Lieferumfang gehören zwei Disketten, die allerdings nur auf das Programmieren in C++ zugeschnitten sind. Es werden nur Include-Files für C++ geliefert; wer als C++-Käufer ein »normales« C-Programm compilieren möchte, dem fehlen die C-Includes des Standard-Compilers.

C++ besteht aus drei weiteren Programmteilen: Einem Steuerprogramm, das wie »LC« die Steuerung der Compilation übernimmt und die zwei Übersetzer aufruft, die ihrerseits ein etwas unübersichtliches C-Listing erzeugen. So würde etwa das vier Zeilen lange »hello.cp« in ein 314zeiliges »_hello.c« übersetzt werden. Danach werden wie gewohnt »lc1« und »lc2« aufgerufen, wahlweise kann das Linken ohne neuen Aufruf erledigt werden. Dazu ist auch eine zusätzliche Library namens »cplus.lib« nötig, alle weiteren Libraries dürften dem Lattice-C-Anwender bekannt sein.

Das erste Handbuch, das die Bedienung des Compilers erläutert, erinnert stark an sein C-Äquivalent. Für den Umsteiger gibt es wenig Neues, da der größte Einfluß auf das Compilat über den C-Compiler genommen werden kann. Wesentlich interessanter ist das Handbuch zur Programmiersprache selbst. Da C++ weithin unbekannt ist, hat man sich entschlossen, dem C++-Neuling eine ausführliche Sprachbeschreibung auf den Weg zu geben. Da dieses Handbuch speziell das objektorientierte Programmieren in C++ behandelt, sind Kenntnisse über C zum Verständnis erforderlich. Für den Umsteiger erübrigt sich so die Anschaffung weiterer Literatur.

Der Pre-Compiler des C++ schlägt mit etwa 300 KByte zu Buche und vergrößert so den Speicherbedarf drastisch. Im Sinne einer erträglichen Compilationszeit stimmen wir den im Handbuch genannten Mindestanforderungen von 1,5 MByte Speicher und einer Festplatte zu. C++ ist damit nur wenigen vorbehalten.

Ob sich diese Sprache zur professionellen Programmentwicklung durchsetzen wird, läßt sich noch nicht beurteilen. Einige aufgetretene Mängel lassen hier Zweifel aufkommen. So ist beispielsweise das Pre-Compiler-Konzept sicher nicht gerade zukunftsträchtig. Mit Sicherheit ist C++ für denjenigen von Interesse, der einen Einblick in die objektorientierte Programmierung bekommen möchte. In diesem Fall bietet sich ein faszinierendes Experimentierfeld – schade ist nur, daß der Preis von fast 800 Mark für den Durchschnittsanwender hoch ist. *(Christian Wolf/so)*

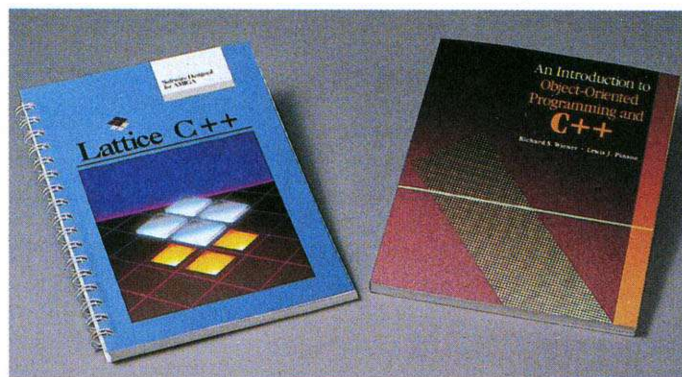


Bild 1. Lattice C++ öffnet dem engagierten C-Programmierer ein faszinierendes Experimentierfeld

Lattice C++ ist zum Preis von ca. 800 Mark erhältlich bei Philgerma, Barerstr. 28, 8000 München, Tel. 089/281228

Assembler im Wettbewerb

Für den Amiga gibt es inzwischen sehr leistungsfähige Assembler. Die Vor- und Nachteile der wichtigsten haben wir gründlich untersucht. Lesen Sie, was Devpac, Profimat und Co. zu bieten haben.

Das Herz des Amiga ist der Mikroprozessor mit der Bezeichnung 68000. Die Vorteile dieses Prozessors sind Schnelligkeit und ein großer Befehlsvorrat. Wenn Sie dem Amiga Beine machen wollen, ist die Programmierung in Maschinensprache der richtige Weg. Da alle Mikroprozessoren jedoch nur die logischen Zustände »Strom fließt« und »Strom fließt nicht« kennen, müssen alle Befehle aus den Zahlen 1 und 0 bestehen. Nun gibt es tatsächlich Programmierer, die Ketten aus diesen Werten »lesen« und auch verstehen. Aber auch diese Gehirnakrobaten arbeiten natürlich mit einem geeigneten Werkzeug, dem Assembler. Dieser übersetzt Befehle, zum Beispiel »RTS« (ReTurn from Subroutine), in eine dem Prozessor verständliche Form.

Für den Amiga werden inzwischen verschiedene Assembler angeboten. Diese unterscheiden sich beispielsweise im Komfort bei der Eingabe der Programme, in der gebotenen Hilfeleistung bei der Fehlersuche und in der Klarheit der Handbücher. Wir haben folgende Kandidaten getestet:
- A68K (Fish-Disk 110)
- CAPE68k (Version 2.0)
- Devpac Assembler (Vers. 2.0)
- Profimat
- Seka

Auf eine Bewertung mit einem entsprechenden Kasten, wie Sie es vielleicht vom AMIGA-Magazin gewohnt sind, haben wir verzichtet. Vielmehr als auf die pure Qualitätskontrolle kam es uns darauf an, herauszustellen, für wen ein bestimmter Assembler die derzeit beste Lösung ist.

Der Purist

Das Handbuch vermittelt den ersten Eindruck vom Seka-Assembler. Diesem Produkt der Firma Kuma liegt eine englische Anleitung bei, die man wohlwollend mit »schön übersichtlich« bezeichnen kann. Auf weitschweifende Ausführungen wurde jedenfalls verzichtet. Die Seka-Neulinge werden einige Zeit benötigen, die im Handbuch angedeuteten Funktionen effektiv einzusetzen.

Auch bei anderen Merkmalen erweist sich der Veteran unter den Amiga-Assemblern als recht eigenwillig. Das Konzept ist grundlegend unterschiedlich zu seinen Konkurrenten.

Das verhältnismäßig kleine Programm beinhaltet Editor, Assembler und Linker. Der Editor enthält die wichtigsten Funktionen für die Erstellung eines Quellprogramms und ist beim Scrollen von Texten nicht gerade der Schnellste.

Dafür dauert der Assemblierungsvorgang nur sehr kurze Zeit. Seka holt sich den Quellcode aus dem Speicher, erzeugt je nach gewählter Arbeitsweise Link- oder Maschinencode und legt diesen auch wieder im Speicher ab. Assemblieren von oder auf die Diskette ist nicht vorgesehen. Die Codepuffer können dann als ausführbares Programm oder Linkmodule auf Diskette gespeichert werden. Mit dem Befehl H (How Big) lassen sich Größe und Adresse der verwendeten Speicherbereiche auf dem Bildschirm ausgeben. Erzeugter Linkcode kann in einen speziellen Linkpuffer kopiert werden.

Ist dieser Puffer nicht leer, so wird der Inhalt mit dem bei weiteren Assemblierungen erzeugten Code verkettet. Mit Hilfe des integrierten Debuggers läßt sich das Programm starten und mit ähnlichen Funktionen wie beim Devpac (siehe unten) auf Fehler untersuchen. Einfacher geht es wirklich nicht.

Trotz der einfachen Bedienung fallen auch dunkle Schatten auf den Assembler. So akzeptiert Seka den Befehl »MOVEA« nicht und weist die Adressierung des Bedingungscoderegisters mit »CCR« zurück. Bei der Gestaltung der Assemblerdirektiven (EQU,ORG und andere) hielt man sich nicht wie die Konkurrenten an den Motorola-Standard. Eine Anpassung fremder Programme ist damit notwendig.

Ein weiterer schwerwiegender Nachteil für die professionelle Programmierung ist die fehlende Möglichkeit zur Einbindung von Quelltexten während der Assemblierung. Die anderen Assembler kennen hierfür die Anweisung »INCLUDE Filename«. Auf deren Disketten findet man auch eine Reihe Include-Files. In diesen Files werden hauptsächlich den Routinen und Variablen des Betriebssystems – und das sind nicht wenig – symbolische Namen gegeben. Assemblerprogramme, die auf das Betriebssystem zurückgreifen, sind so wesentlich einfacher zu programmieren.

Komfort mit Makros

Alle vorgestellten Kandidaten sind Makroassembler. Makros sind durch »Name MACRO« und »ENDM« eingeschlossene Textzeilen im Quellprogramm. Taucht bei einer Übersetzung nach der Definition des Makros dessen Name im Quelltext auf, so ersetzt der Assembler den Namen durch die entsprechenden Textzeilen. Das Makro INC im folgenden Beispiel simuliert einen Inkrementbefehl:

```
INC: macro  ; Definition
    ADDQ #1,\1
ENDM
INC D0    ; Aufruf
```

Der Assembler setzt beim Aufruf an die mit <\> und einer Ziffer gekennzeichneten Stellen den entsprechenden Parameter ein.

Eine leistungsfähige Makroverwaltung ist eine unschätzbare Hilfe für den Programmierer. Es lassen sich damit Befehlsfolgen aufbauen, die ähnlich komfortabel sind wie Funktionsaufrufe in höheren Programmiersprachen. Beispiel:

```
MOVEWINDOW: macro
 move.l \1,a0
 move.l \2,d0
 move.l \3,d1
 move.l IntuitionBase,a6
 jsr _MoveWindow(a6)
endm
```

Nach der Definition des Makros wird dann mit der Anweisung »MOVEWINDOW Window, #10, #10« ein Intuition-Fenster um jeweils zehn Punkte nach rechts und nach unten verschoben. Das einmalige Definieren eines Makros entlastet fortan den Programmierer von der Überlegung, in welchem der Register die Parameter der Betriebssystemroutinen stehen müssen. Während sich der Amiga unter Umständen bis zum nächsten Reset verabschiedet, wenn Seka innerhalb des aufgerufenen Makros einen weiteren Aufruf findet, können seine Konkurrenten Makros verschachteln.

Mit der bedingten Assemblierung ist es möglich, den erzeugten Code ohne Änderung des Quellprogramms zu variieren. Eine Anwendung dafür ist das Einfügen von Debugging-Hilfen bei Testversionen:

```
Upro1:
IFNE DEBUG
  lea Text,a0  ;
  jsr StringOut
; Text ausgeben
ENDC
  ...
  rts  ; Ende von Upro1
IFNE DEBUG
Text: dc.b "Bin im
Upro Eins",0
ENDC
```

Der Pseudo-Opcode (Pseudo-Operationscode = kein echter 68000-Operationscode) »IFNE« überprüft den Wert des Symbols DEBUG. Ist dieser ungleich Null, werden die Zeilen bis »ENDC« übersetzt. Eine solche Technik kann sinnvoll sein, um bei auftretenden Fehlern den Programmablauf am Bildschirm zu verfolgen. Funktioniert das Programm einwandfrei, so wird durch Setzen von DEBUG auf Null die Debugginghilfe nicht mehr assembliert.

Während Seka praktisch nur die Pseudo-Ops IFEQ und IFNE kennt, arbeiten die anderen Testkandidaten auch hier nach dem Motorola-Standard, der als Parameter für »IF« fast alle Bedingungen vorsieht, die auch der 68000 in seinen Branchbefehlen verwendet.

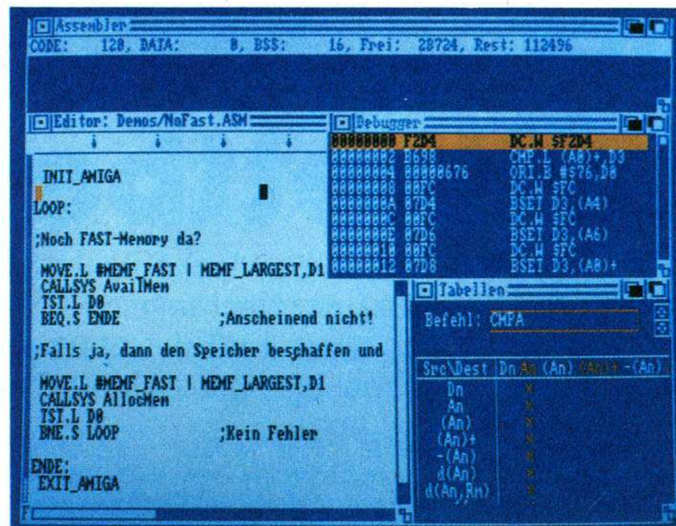


Bild 1. Die einzelnen Module des Profimat-Assemblers sind natürlich auch im Multitasking-Betrieb lauffähig

Der Seka-Assembler ist wegen der einfachen Bedienung sehr gut für kurze Programme oder erste Kontakte mit Prozessor und Betriebssystem geeignet. Ohne große Wartezeiten kann man mit diesem Werkzeug am Objekt lernen. Sobald man sich mit dem Amiga besser auskennt und größere Projekte in Maschinensprache realisieren will, ist der Umstieg auf einen leistungsfähigeren Konkurrenten unvermeidlich.

Profi-Assembler

Der »Profimat« von Data Becker ist ein kompaktes Entwicklungssystem, das dem Assemblerprogrammierer einen übersichtlichen Aufbau und leichte Bedienung verspricht. Der Lieferumfang des Paketes enthält neben einer Diskette ein über 140 Seiten umfassendes Handbuch, das durch eine ausgezeichnete Struktur überzeugt. Es führt den Leser zu Beginn in den allgemeinen Umgang mit Assemblern ein und beschreibt anschließend die Arbeitsweise des Profimat.

Nach Kenntnisnahme der Installationshinweise und Anmerkungen zur Bedienung der Benutzeroberfläche ist der Leser sehr schnell in der Lage, sein erstes Maschinenspracheprogramm einzugeben und auszuprobieren. Durch dieses erste Erfolgserlebnis motiviert, wird der engagierte Assembler-Fan das Buch gar nicht mehr aus der Hand legen.

Der Hauptteil der Dokumentation beinhaltet die Funktionsbeschreibung aller Menüpunkte und Befehle mit detaillierten Erklärungen und Beispielausdrucken. Zusätzliche Kapitel weisen in die Bedienung von Editor und Debugger ein. Eine Übersicht der verwendbaren Kommandos, der Assemblerbefehle und Fehlermeldungen rundet die Dokumentation ab.

Das eigentliche Programm besteht aus den vier Modulen Editor, Assembler, Debugger und Reassembler (Bild 1). Ein Editor ist im Prinzip eine »abgespeckte« Textverarbeitung. Einfache Funktionen (Tabelle 1) ermöglichen das Erstellen (oder Editieren) eines sogenannten Quellprogramms. Der Profimat-Editor wird bis auf kleine Ausnahmen durch Pull-Down-Menüs bedient und paßt sich somit dem Erscheinungsbild an, das sich auch schon bei anderen Amiga-Programmen bewährt hat. Die maximale Breite einer Zeile beträgt 127 Zeichen. Dies reicht in der Regel vollkommen aus.

Die Leistung überzeugt

Der Programmierer kann nach der Fertigstellung des Quelltextes durch Wahl des entsprechenden Menüpunktes das Programm assemblieren und starten. Der Profimat versteht den vollen Motorola-68000-Befehlssatz und erlaubt auch Alternativbefehle wie beispielsweise XOR und AND statt EOR und ANDI (Tabelle 2). Eine Reihe von Assemblerdirektiven (Pseudo-Opcodes) vereinfacht die Programmentwicklung (Tabelle 3).

So ist eine Include-Anweisung für das Einbinden von öfter verwendeten Variablendefinitionen oder Programmteilen vorhanden. Im Gegensatz zum Seka-Assembler von Kuma enthält der Profimat die Include-Dateien mit den Struktur- und Symboldefinitionen für das Amiga-Betriebssystem. Kontrollstrukturen wie REPEAT... UNTIL erleichtern die wiederholte Assemblierung einzelner Programmteile. Die bedingte Assemblierung mit IF, IFD, IFND erlaubt eine Modifizierung des Maschinencodes durch Angabe bestimmter Parameter beim Start der Übersetzung. Damit kann zum Beispiel derselbe Quellcode auf verschiedene Computer angepaßt werden. Mit der Definition von Makros lassen sich spezielle Befehlsfolgen nur durch die Angabe des Makronamens in den Quelltext einfügen. Mit SLABEL kann der Programmierer sich eine Bibliothek mit Symbolen anlegen, die dann durch ILABEL in weiteren Programmen verfügbar gemacht werden können. Der Profimat unterstützt außerdem lokale und redefinierbare Variablendefinitionen.

Sämtliche Zeichen im Namen der Labels sind signifikant. Wer einmal längere Zeit einen Programmfehler gesucht und als Ursache letztlich eine Übereinstimmung der signifikanten Stellen unterschiedlicher Sprungmarken erkannt hat, weiß dies zu schätzen.

Beim Assemblieren eines Textes stellt der Profimat ein reichhaltiges Potential an Funktionen zur Verfügung, die sogar die Ausgabe einer übersichtlichen Cross-Reference-Tabelle ermöglichen. Sehr positiv sind auch die sogenannten Hilfstabellen zu bewerten. In ihnen sind alle 68000-Befehle mit ihren zulässigen Adressierungsarten und alle Betriebssystem-Routinen mit den zugehörigen Parametern abrufbar.

Neben all diesen Pluspunkten wird das Leistungskonto des Profimat allerdings durch die recht langsame Assemblierung belastet. Unverständlich ist auch die fehlende Implementierung der in anderen Assemblern verbreiteten Befehle EVEN und CNOP. Dadurch wird die Übernahme von »fremden« Quelltexten unnötig erschwert. Das Handbuch gibt zwar Hinweise, wie man diese Befehle durch Makros selbst definieren kann, eine direkte Implementation wäre aber weniger umständlich gewesen.

Selbst wenn Sie kein Anfänger bei der Programmierung in Maschinensprache sind, müssen Sie den Begriff »Reassembler« nicht unbedingt kennen.

Editier-Befehle des Profimat

Blockbefehle	
BA	markiere Blockanfang
BC	kopiere Block
BE	markiere Ende
BH	hebe Markierung auf
BK	lösche Block
BM	verschiebe Block
BP	füge Block ein
BV	zeige Block an
BX	schneide Block aus
Cursorsteuerung	
CL	Cursor nach links
CO	Cursor nach oben
CR	Cursor nach rechts
CU	Cursor nach unten
C(x,y)	setze Cursor auf Position x,y
CN	Cursor an den Anfang der nächsten Zeile
CV	Cursor an den Anfang der vorherigen Zeile
Löschen	
LA	lösche bis Zeilenanfang
LE	lösche bis Zeilenende
LL	lösche das Zeichen links vom Cursor
LR	lösche das Zeichen rechts vom Cursor
LZ	lösche Zeile
Tabulatorbefehle	
TL	Tabulator löschen
TS	Tabulator setzen
Sonstige Befehle	
Z Zeichenkette	Zeichenkette einfügen
ZH Zeichenkette	Zeichen hinter aktueller Zeile einfügen
ZV Zeichenkette	Zeichenkette vor momentaner Zeile einfügen

Tabelle 1. Der Editor des Profimat wird durch Menü- und einfach zu erlernende Tastatur-Kommandos gesteuert

Profimat-Alternativen

Motorola-Standard	Alternative
ADDA, ADDI	ADD
SUBA, SUBI	SUB
CMPA, CMPI	CMP
ANDI	AND
EORI, EOR	XORI, XOR
EOR	XOR
ORI	OR
MOVEA	MOVE

Tabelle 2. Der Profimat bietet bei den Befehlsnamen Alternativen zum Motorola-Standard

Prinzipiell hat ein Reassembler dieselben Funktionen wie ein Disassembler: Er wandelt im Speicher befindliche Maschinencodes wieder in Assemblersprache um. Während die Ausgabe eines Disassemblers in der Regel nur für die Erzeugung von Assemblerlistings oder der Betrachtung kurzer Programmroutinen dient, können vom Reassembler erzeugte Dateien wieder als Eingabe für den Assembler genutzt werden. Zu diesem Zweck wird für jede Adresse, auf die in irgendeiner Weise von den Befehlen des zu reassemblierenden Bereichs zugegriffen wird, eine symbolische Bezeichnung generiert.

Der im Profimat integrierte Reassembler unterscheidet weitgehend zwischen Daten und Programmroutinen. Er erkennt direkt angesprungene Unterroutinen, merkt sich deren Ende und erklärt alle nicht auf diese Weise durchlaufenen Schritte als Daten. Indirekt angesprungene Befehle können so natürlich nicht als Programm erfaßt werden.

Der Debugger kann durch seinen geringen Leistungsumfang nicht überzeugen. So hilft er dem Anwender zwar, die beliebten Guru-Meditation-Fehlermeldungen abzufangen, ein angeschlossener Drucker läßt sich damit jedoch nicht aktivieren. Der Einzelschrittmodus und die 68020-Einzelschritt-Emulation funktionieren zufriedenstellend. Ein Breakpoint kann allerdings nicht im ROM-Bereich positioniert werden.

Trotz der wenigen Schwachpunkte erhält sowohl der interessierte Einsteiger als auch der professionelle Programmierer mit dem Profimat ein Werkzeug, das mit einigen Funktionen konkurrenzlos ist. Der hohe Bedienungskomfort und die Funktionsvielfalt garantieren kurze Entwicklungszeiten für Assemblerprogramme. Die zahlreichen Anpassungsmöglichkeiten erlauben auf eine Aufgabe bestens zugeschnittene Problemlösungen. Durch die Makrodefinitionen, die Unterstützung der 68010-Befehle und die Angabe von lokalen und redefinierbaren Variablen ist der Profimat auch für die Realisation von komplexen Programmstrukturen ideal geeignet. Der von vielen Data Becker-Produkten bekannte niedrige Preis vereinfacht die Entscheidung für dieses Produkt, das sicher jeden Assembler-Profi überzeugen kann.

Ein Umhang mit vielen Vorteilen

Der Makro-Assembler von Metacomco ist eine der ältesten Programmiersprachen für den Amiga. Eine komfortable Benutzeroberfläche bietet der CAPE68K (Version 2.0), der zu dem Metacomco-Assembler voll kompatibel ist.

Inovatronics zeigt mit ihrem »Complete Assembler Programming Environment« (CAPE), daß sich immer Verbesserungen anbringen lassen. Diese Firma ist bekannt für ihre einfallsreichen Programmier-Tools. Als Beispiel sei PowerWindows (siehe auch Seite 101 angeführt, mit dem es ohne große Text-Tipperei möglich ist, Fenster- und Menü-Strukturen zu erstellen.

CAPE bedeutet in der englischen Sprache »Umhang«, und entsprechend ist dieses Paket auch aufgebaut. Um den Assembler, wie er vielen bereits bekannt ist, wurde eine Schale gelegt, eine leicht zu bedienende und angenehme Benutzeroberfläche. CAPE besteht neben dem Assembler aus einem Editor und einer Einrichtung zur Unterstützung des EXEC-Message-Systems. Der Assembler läßt sich über das sogenannte ARexx-Interface durch einen externen Kommando-Interpreter steuern.

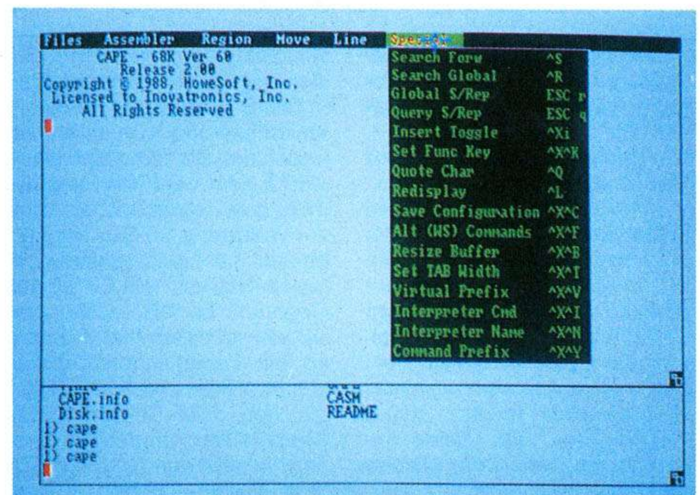


Bild 2. Die Menüs der CAPE68k zeigen auch die Shortcuts

Vielseitiger Editor

Der Editor des Pakets (Bild 2) zeichnet sich durch zwei umschaltbare Befehlstasten-Sätze aus. Entsprechend Ihrer Gewohnheit können Sie zwischen den Steuertasten von Wordstar und denen von Micro-Emacs wählen. So müssen Sie sich nicht umgewöhnen, wenn Ihnen die Arbeit mit einem dieser Programme vertraut ist. In beiden Modi enthält der Editor alle nötigen Funktionen, wie zum nächsten Wort springen oder löschen bis zum Ende der Zeile.

Der Assembler läßt sich aus dem Editor starten. Leider ist der Aufruf des Linkers von dort nicht vorgesehen. Zum Testen der Programme ist es noch notwendig, ein CLI-Fenster geöffnet zu haben, aus dem der Linker und das Programm aufgerufen werden können.

Pseudo-Opcodes

Assemblerbefehl	Bedeutung
ALIGN.x	Sicherstellen, daß die nachfolgenden Befehle an geraden Adressen stehen
BSS	Beginn eines Block-Storage-Segments
CODE	Beginn eines Code-Segments
DATA	Beginn des Daten-Segments
DC.x	Datendefinition von Daten beziehungsweise Werten, die als Konstanten im Speicher abgelegt werden
DS.x n[,Füllwert]	Reservierung eines Speicherbereichs der Länge »n« und dem Inhalt »Füllwert«
END	Obligatorischer Abschluß des Quelltextes
Name: Wert oder Name EQU Wert	Dem Symbol »Name« wird ein Wert zugewiesen
Name EQUR String	Einem Symbol wird eine Zeichenkette zugewiesen
IBYTES Datei[,Länge]	Daten, die auf dem externen Massenspeicher vorliegen, werden an der Stelle, an der dieser Befehl steht, eingefügt
IF; IFD; IFND	Bedingte Assemblierung
INCLUDE Datei	Quelltextdatei einbinden
INPUT [Meldung,] Variable	Die »INPUT«-Funktion hat dieselbe Wirkung wie die normale Definition eines Speicherbereichs mit » = « oder »EQUR«, nur daß Sie den Wert während des Übersetzungsvorgangs festlegen können
LIST; NOLIST; PAGE	Druckerkommandos
MACRO	Makrodefinition
ORG Adresse	Startadresse des Programms
REPEAT; UNTIL	Wiederholte Assemblierung
SLABEL; ILABEL	Labelbibliotheksfunktionen

Tabelle 3. Die Pseudo-Operanden des Profimat überzeugen

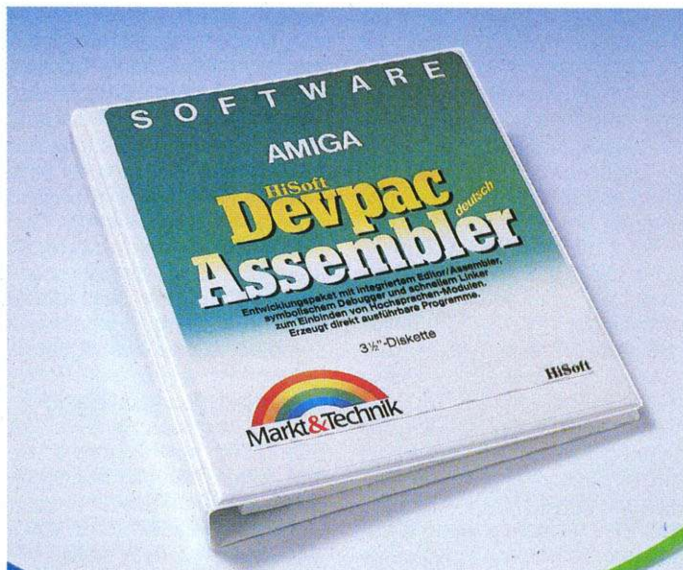


Bild 3. Der Devpac bietet in der Version 2.0 noch mehr Vorteile

CAPE bietet auch solche Extras wie Funktionstasten-Belegung, die das Arbeiten mit einem Editor zur Freude machen. Alle Funktionen sind auch über Menüs erreichbar.

Der Assembler ist teilweise kompatibel zum Metacomco- und dem Devpac-Assembler. Während Devpac kleinere Ungenauigkeiten vergibt, legt CAPE Wert auf genaue Einhaltung der Motorola-Syntax. Devpac wandelt ein »btst.w #10,Adresse« in ein »btst.b #2,Adresse« um, beide Instruktionen haben die gleiche Wirkung. Im Gegensatz zum Seka-Assembler unterstützt CAPE Include-Files. Eine weitere Eigenschaft, die CAPE zu einem interessanten Werkzeug macht: Das Programm springt nach jedem Fehler an die Stelle im Editor, an der dieser aufgetreten ist.

Die interessanteste Neuerung bei der Version 2.0 des CAPE ist sicherlich die Fähigkeit, das Programm während des Assemblierens zu optimieren. Bisher wurden nur Kleinigkeiten an Programmen verbessert. Sprünge wurden als »Short Branches« ausgewiesen, wenn die Distanz es zuließ. CAPE geht in der Optimierung allerdings einige Schritte weiter. Wenn es möglich ist, werden die »Quick«-Versionen der Befehle eingesetzt. Dazu verwendet CAPE ein sogenanntes Trash-Register. Wenn im Programm der Befehl »add.l #120,d0« auftaucht, wandelt der Assembler das um in die Befehlsfolge »moveq.l #120, Trash-Register« und »add.l Trash-Register,d0«. Das spart 2 Byte im fertigen Programm. Ein »add.l #0,d0« wird ganz aus dem Code gelöscht, da es sowieso keine Konsequenzen für die Programmausführung hat.

CAPE unterstützt bedingte Assemblierung. Dazu stehen neben den Standardfunktionen »IFEQ« und »IFNE« weitere numerische Vergleichsoperatoren zur Verfügung. »IFGE« fragt ab, ob der Ausdruck größer oder gleich Null ist, »IFLE« ist das Gegenstück. Mit »IFD« und »IDND« kann ermittelt werden, ob ein Label schon definiert wurde. »IFC« und »IFNC« vergleichen zwei Strings. Jedes bedingte Programmteil wird abgeschlossen durch ein »ENDC« oder »ENDIF«, die bei CAPE synonym zu verwenden sind.

Die bedingte Assemblierung wird sehr hilfreich bei Verwendung mit Makros. CAPE stellt Befehle für Makros mit bis zu zehn Parametern und 8 KByte Länge zur Verfügung. Die Parameter werden übergeben durch »\1« bis »\0«. »\@« ist ein besonderer Operator in Makros. Für diese Zeichenfolge wird in jedem Makro eine andere Zahl eingesetzt. Das verhindert, daß Labels in Makros doppelt vorkommen.

Seit einiger Zeit schon gibt es Programmierer-Teams, die durch gute Einfälle den Markt bestimmen. Viele dieser Programmierer veröffentlichen ihre Ergebnisse in den Public Domain-Reihen, so auch die »Software Destillery«. Von dieser Gruppe stammt der Linker »BLink«, der mit dem CAPE ausgeliefert wird. Dieser Linker versteht mehr »Kommandos« als der Metacomco-Linker »ALink«. Speziell die SMALL-CODE-Option und die Alternative, ob das Programm in den Chip- oder in den Fast-Speicher geladen werden soll, fehlen bei ALink.

Mit CAPE ist es ebenso möglich, ausführbare Programme zu erstellen, ohne den Linker zu benutzen. Dazu muß der Befehl »EXEOBJ« in den Sourcecode eingefügt werden. CAPE speichert nun direkt ausführbare Programme ab. Viele Fähigkeiten des Linkers lassen sich so nicht nutzen. EXEOBJ ist eher zum schnellen Ausprobieren von Programmteilen gedacht.

Ein weiterer Standard, der sich wie BLink über die Public Domain etablieren konnte, ist der »ARexx«-Standard von William Hawes. Zur Kommunikation verschiedener Programme miteinander ist ARexx gedacht. So kann ein Programm entwickelt werden, das CAPE von »außen« steuert. Alle Steuerfunktionen, die auch von der Tastatur verwendet werden können, lassen sich durch ein Programm betätigen. CAPE liefert in diesem Falle keine Fehlermeldung auf dem Bildschirm, sondern sendet dem Kommando-Interpreter einen Fehler-Code. Der ARexx-Standard wird sicher fortgeschrittene Programmierer interessieren.

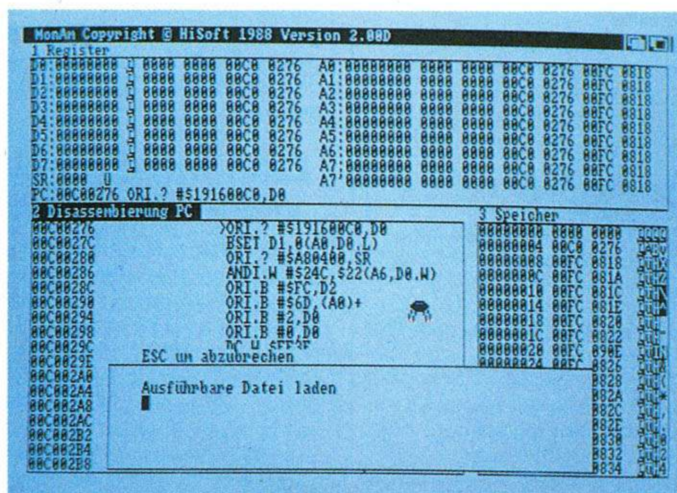


Bild 4. Debuggen mit dem Devpac – finden Sie die Wanzen?

Durch dieses Interface können Programme geschrieben werden, die CAPE um zusätzliche Funktionen erweitern. Ein externes Programm könnte durch eine Funktionstaste dazu gebracht werden, das gerade vom Assembler produzierte File zu linken und auszuführen. Sehr viel mächtigere Funktionen sind vorstellbar.

Das Paket CAPE wird mit einem englischsprachigen Handbuch ausgeliefert. Obwohl dieses Manual nicht als Lehrbuch für 68000er-Maschinensprache ausgelegt ist, findet sich hier noch einmal eine Übersicht der Assembler-Befehle. Die Editor-Steuertasten sind gut erklärt und übersichtlich in Tabellen aufgeführt. Die Verwendung von Makros und der bedingten Assemblierung ist kommentiert und mit Beispielen illustriert. Auch das ARexx-Interface ist gut dokumentiert, so daß erfahrene Programmierer ihre eigenen Kommando-Interpreter anhand der im Handbuch gebotenen Informationen erstellen können.

Wie dieser Assembler sich im Markt etabliert, hängt sicher auch davon ab, wie schnell Inovatronics einen interessanten Kommando-Interpreter nachschiebt. Unser Eindruck von dem »alten« Assembler im neuen Gewand ist einer der besten.

Der PD-Assembler

Das Bonbon auf der Fish-Disk 110 ist der Assembler A68K. Mit diesem Zwei-Pass-Assembler können Sie in Verbindung mit dem Linker »Blink« (ebenfalls auf der Diskette) vernünftig arbeiten. Alle Anweisungen sind auf der Diskette dokumentiert. Die wenigen Nachteile des Assemblers:

- Fehlermeldungen lassen sich noch nicht auf eine Datei umleiten, sondern erscheinen auf dem Bildschirm.
- Dateinamen im Include-Verzeichnis müssen durch Kommata getrennt werden.

Alles halb so wild. Die Aufzählung der Leistungsmerkmale ist um einiges länger:

- Sprungmarken können bis zu 127 Byte lang sein.
- Der Assembler erlaubt Include-Dateien.
- Sprung- und Move-Befehle werden optimiert.
- A68K kennt Makros.
- Sie haben Kontrolle über Fast- und Chip-RAM.

Für geübte Programmierer ist der gesamte Quellcode von A68K interessant, der sich

ebenfalls auf der Diskette befindet. Was dem Assembler fehlt, ist die Amiga.lib und vorgefertigte Include-Dateien.

Viele Listings lassen sich durch ein paar kleine Änderungen an den Assembler anpassen. Wir empfehlen diesen Assembler daher jedem, der sich mit der Materie vertraut machen möchte. Immerhin besitzen Sie dann ein Werkzeug, um erste Schritte in Maschinensprache zu wagen.

Das Komplettpaket: Devpac 2.0

Der Devpac (Development Package) ist ein komplettes Entwicklungspaket. Es besteht aus Editor, Assembler, Linker und Debugger. Alle Funktionen können in der neuen Version 2.0 vom Editor aus gesteuert werden. Zusätzlich gehört eine Stand-alone-Version des Assemblers zum Lieferumfang, die ohne Editor-Steuerung auskommt. Bild 3 zeigt die Menüs des Editors. Zu einigen existieren weitere Untermenüs. »Programm« ist der Teil, der alle Funktionen integriert.

Der Editor ist über Shortcuts (Tastenkürzel) oder mit der Maus zu steuern. Er arbeitet sehr schnell und bietet hervorragende Leistungsmerkmale.

Wenn Sie vom Editor aus den Menüpunkt »Assembler« aufrufen, fällt ein wichtiger Punkt ins Auge: Es kann in den Speicher assembliert werden. Das Programm ist nun direkt ausführbar. Taucht bei einem Testlauf des Programms im Speicher ein Fehler auf, springt der Cursor im Editor in die fehlerhafte Zeile. Dieses ermöglicht schnelles und komfortables Entwickeln und Bearbeiten eines Programms. Wichtige Bedingung ist allerdings, daß Include-Files (die der Devpac selbstverständlich einlesen kann) von einer Festplatte oder besser aus der RAM-Disk geladen werden. Ein gut ausgebauter Speicher und eine resetfeste RAM-Disk sollten also in jedem Fall vorhanden sein.

Die SECTION-Anweisung des Devpac ermöglicht die Positionierung von Daten entweder im Chip-Memory oder im Fast-Memory. Wie Sie vielleicht wissen, müssen beim Amiga Daten, die von den Custom-Chips verarbeitet werden sollen, unbedingt im Chip-Memory liegen.

Makros konnten schon in der Version 1.1 des Devpac beliebig tief verschachtelt sein und dürfen sich auch selbst aufrufen (Rekursion). Neu ist in der Version 2.0, daß Devpac jetzt bis zu 36 Parameter akzeptiert und Makroaufrufe sich über mehrere Zeilen erstrecken können. Texte in spitzen Klammern können in der neuen Version Leerzeichen enthalten. Lokale Labels sind nun möglich, womit Makros besser lesbar und zu handhaben sind als mit der »n$«-Form, die alternativ weiterhin erlaubt bleibt.

Mit MonAM auf Fehlersuche

Der Assembler kann sofort ausführbaren Code erzeugen, es sind aber auch »linkbare« Module zu erstellen. Diese sind dann mit dem Linker »BLINK« mit anderen Modulen zu binden, auch Module von Hochsprachen sind mit dem Linker selbstverständlich einzubinden.

Der Assembler arbeitet nicht nur sehr schnell, er bietet auch einige echte Bonbons:
- bedingtes Assemblieren,
- Repeat-Schleifen,
- Zugriff auf Assembler-Variablen,
- Anweisungen zum einfachen Aufbau von Datenstrukturen.

Ein leistungsfähiges Werkzeug zur Fehlersuche bietet das Devpac-Paket mit dem Debugger »MonAm«. Rufen Sie aus dem Editor heraus den Punkt »Debuggen« auf, erscheint ein dreigeteiltes Ausgabefenster (Bild 4). In den drei Fenstern sind das Reassembler-Listing, alle Register und ein Speicherauszug in ASCII und hexadezimal dargestellt. Jedes der Fenster kann per Tastendruck auf volle Bildschirmgröße gezogen werden. Nun ist in Einzelschritten der Programmablauf nachzuvollziehen, Unterprogramme können ignoriert werden, Breakpoints sind einfach zu setzen. Nützlich ist auch der History-Buffer, in dem die letzten fünf Schritte mit allen Registerwerten gespeichert sind.

Der Devpac 2.0 ist mit dem Profimat zusammen sicher die beste Alternative, wenn Sie intensiv Programme in Assembler entwickeln wollen. Ob sich der CAPE in Deutschland durchsetzen wird, ist fraglich.

(Markus Zietlow/ Ingolf Krüger/ Michael Göckel/rs)

Programmi

Viele Programmierer halten C für die interessanteste Sprache auf dem Amiga. Kein Wunder – bietet sie doch bei rasanter Geschwindigkeit allen Komfort einer Hochsprache. Zudem haben Sie leichten Zugriff auf die Betriebssystemroutinen.

Wollen Sie mit dieser anspruchsvollen Sprache Ihren Amiga programmieren? Dann ist dieser Kurs genau richtig für Sie. Er richtet sich in erster Linie an alle C-Neulinge, die gerne die Interna des Amiga kennenlernen möchten. Danach schöpfen Sie dessen so oft gerühmte Fähigkeiten noch besser aus.

Wir werden im folgenden nur kurz auf allgemeine C-Grundlagen eingehen und mehr Wert auf die Programmierung von Amiga-typischen Anwendungen legen. Umsteigern von anderen Programmiersprachen kommt diese Vorgehensweise sehr entgegen. Wer allzu große Verständnisschwierigkeiten hat, sollte sich mit Hilfe eines Grundlagenbuchs zu C die nötigen Kenntnisse aneignen. Hinweise zu geeigneter Literatur finden Sie auf Seite 147. Außerdem war in der Ausgabe 11/87 bis 4/88 des Amiga-Magazins ein C-Grundlagenkurs enthalten. Wer diese Hefte hat, findet dort eine kurze und verständliche Einführung, die für diesen Kurs vollkommen ausreichend ist.

Vorläufer der Programmiersprache C, wie wir sie heute kennen, waren »BCPL« und später »B«. Anfang der 70er Jahre wurde die Sprache B von Dennis M. Ritchie und Brian W. Kernighan überarbeitet und erweitert. Es entstand C, eine hardware- und betriebssystemunabhängige Compilersprache.

Compilersprache bedeutet, daß der Quellcode eines Programms vor dem ersten Start übersetzt werden muß – im Gegensatz zu Sprachen wie Basic, die gewissermaßen »simultan« übersetzen (sogenannte Interpreter).

Tempo mit kleinem Befehlssatz

Der C-Compiler ist – da der Sprachumfang von C relativ klein ist (siehe Tabelle 1) – sehr schnell, so daß Sie nicht allzu lange auf diese Übersetzung warten müssen. Beim Compilieren wird ein sehr kompakter Code erzeugt. Die Ausführungsgeschwindigkeit der Programme ist so hoch, daß selbst in zeitkritischen Fällen oft auf eine Programmierung in Assembler verzichtet werden kann.

C bietet allen Komfort einer Hochsprache, erlaubt aber gleichzeitig eine maschinennahe Programmierung, wie zum Beispiel Bitmanipulationen. Darüber hinaus stehen dem Programmierer Kontrollstrukturen zur Verfügung, die strukturiertes Programmieren unterstützen. C ist eine universell einsetzbare Sprache, die sich sogar zur Entwicklung von Betriebssystemen eignet. Ein großer Teil des Amiga-Betriebssystems ist, neben BCPL, in C formuliert. Daher ist die Nutzung der Systemroutinen – ganz anders als von Basic aus – besonders einfach. Nicht umsonst bezeichnet man C als die »Muttersprache« des Amiga.

Wir wollen uns einige dieser Routinen etwas genauer ansehen. Das ganze Betriebssystem können wir natürlich nicht durchleuchten, auch ein noch so dickes Sonderheft würde dafür nicht genügend Platz bieten. Wir beschränken uns daher auf wichtige Gebiete wie Intuition, Grafik und AmigaDOS.

Der Aufruf einer Systemroutine setzt meist die Initialisierung einer oder mehrerer, eventuell recht komplizierter Strukturen voraus. Gerade diese Tatsache schreckt viele C-Anfänger ab, tiefer ins System vorzudringen. Um Ihnen den Einstieg in dieses Gebiet zu erleichtern, haben wir in einem speziellen »Headerfile« einige Routinen vordefiniert, die Ihnen die lästige Initialisierung der entsprechenden Strukturen abnehmen.

Alle diese Routinen werden wir Stück für Stück erarbeiten, so daß uns gegen Ende des Kurses ein wertvolles Werkzeug zur Verfügung steht: »Tool.h« (Listing 1). Falls Sie schon eine Tool.h-Version besitzen (beispielsweise von einer der Programmservice-Disketten): bitte Vorsicht! Wir verwenden hier die Version 2.0, für die einiges geändert wurde. Am besten vergleichen Sie beide Listings, fügen die neuen Routinen ein und nehmen die entsprechenden Änderungen vor.

Der folgende Kurs enthält viele Beispielprogramme. Diese wurden mit dem Aztec C-Compiler 3.40a und 3.6a getestet, sollten sich aber auch problemlos mit dem Lattice C-Compiler übersetzen lassen. Hierfür müssen Sie vor allem das Aztec-Includefile »functions.h« durch die Lattice-»Prototypes« ersetzen.

Die Übersetzungsanweisungen lauten, wenn nicht ausdrücklich etwas anderes vereinbart ist:

```
cc name.c +L
ln name.o -lc32
```

Für »name« setzen Sie bitte jeweils den Namen des entsprechenden Programms ein.

Alles auf einmal: Multitasking

Es ist sicher von Vorteil, wenn Sie sich für diesen Kurs eine spezielle Diskette anlegen. Sichern Sie dort alle Beispiele und schaffen Sie sich so ein recht umfangreiches »Unterroutinen-Nachschlage(lade)werk«.

Bevor wir mit den ersten Beispielen beginnen, sollten Sie

Der Sprachschatz von C

Schlüsselworte	Bedeutung
auto	Speicherklasse, blockbezogen
break	Schleifenabbruch
case	Teil der switch-Anweisung
continue	nächste Wiederholung einer Schleife erzwingen
char	Datentyp »Zeichen«
default	Teil der switch-Anweisung
do	Schleife mit Prüfung nach Durchlauf
double	Variablentyp, doppelt genaue Gleitkommazahl
else	leitet Alternativblock der if-Anweisung ein
enum	Datentyp, Aufzählungstyp
extern	Speicherklasse, globale Variable
float	Datentyp, Gleitkommazahl
for	Schleife mit Prüfung vor Durchlauf
goto	Sprunganweisung
if	Entscheidungsanweisung
int	Datentyp, ganze Zahl
long	Datentyp, erweiterte ganze Zahl
register	Speicherklasse wenn möglich in Register
return	Funktionsende, kann einen Wert zurückgeben
short	Datentyp, ganze Zahl
sizeof	liefert die Größe des Operanden
static	Speicherklasse, Lebensdauer über Blockgrenzen
struct	Datentyp, Struktur
switch	Anweisung zur Auswahl verschiedener Möglichkeiten
typedef	zur Definition eigener Datentypen
union	Datentyp, Union
void	Typ einer Funktion ohne Returnwert
while	Schleife mit Prüfung vor Durchlauf

Tabelle 1. Alle Schlüsselworte der Programmiersprache C. Diese Namen dürfen nicht für andere Zwecke verwendet werden. Sie haben in C eine feste Bedeutung.

eren mit C

sich klar machen, in welcher Umgebung Sie programmieren und welche Regeln Sie einhalten müssen.

Als Programmierer auf dem Amiga ist es nicht nur notwendig, sein eigenes Programm korrekt zu formulieren, man muß darüber hinaus darauf achten, daß dieses auch mit anderen, eventuell gleichzeitig ablaufenden Programmen nicht in Konflikt gerät. Das Betriebssystem des Amiga ist, wie Sie wissen, multitaskingfähig. Doch was bedeutet das?

Multitasking ist – das Wort deutet es an – das gleichzeitige Abarbeiten mehrerer Aufgaben (»Tasks«), in unserem Fall mehrerer Programme. Natürlich kann ein Computer, solange er mit nur einem Prozessor arbeitet, immer nur ein Programm gleichzeitig bearbeiten. Ein Blick in das Gehäuse bestätigt: auch der Amiga verfügt nur über einen Prozessor, wenn auch über einen recht schnellen »68000er« von Motorola.

Überzeugender Simulant

Daraus folgt, daß der Amiga nicht in der Lage ist, mehr als ein Programm gleichzeitig abzuarbeiten – er simuliert nur, wenn auch überzeugend. Das scheinbar gleichzeitige Abarbeiten wird erreicht, indem der Prozessor im schnellen Wechsel nacheinander einen kleinen Teil jedes der anstehenden Programme bearbeitet. Die Verwaltung aller Tasks übernimmt der Betriebssystemteil »Exec«. Im Normalfall erhält jeder Task die gleiche Menge an Rechenzeit, jedoch ist das Setzen von Prioritäten von –128 bis +127 erlaubt. Je höher die Priorität, desto mehr Rechenzeit erhält ein Task und umso schneller ist er daher abgearbeitet.

Zwei Programme gleicher Priorität teilen sich also die Leistung des Prozessors. Es taucht die Frage nach dem Sinn des Ganzen auf, denn ganz offensichtlich laufen die Programme dann nur noch mit halber Geschwindigkeit – oder?

Die Anwort lautet »nein«. Die Erklärung klingt kurios: Die Hauptttätigkeit eines Prozessors ist, so seltsam es auch erscheinen mag, warten! Ständig wartet er mit rasanter Geschwindigkeit auf irgendwelche Ereignisse, zum Beispiel auf Bestätigung eines Requesters oder auf die Auswahl aus einem Menü, er wartet in Editoren oder Textprogrammen auf Tastendrücke (auch wenn der Anwender über noch so flinke Finger verfügt), er wartet in Zeichenprogrammen auf das Zeichnen des nächsten Punktes, er wartet beim Drucken auf die Mechanik des Druckers oder bei Laden und Speichern auf die Rückmeldung der Diskettenstation und, und, und – eine Warteliste ohne Ende.

Diese Wartezeit wird nun zur Erledigung anderer Aufgaben genutzt. Anstelle auf ein Ereignis zu warten, arbeitet der Prozessor andere Tasks ab. Die Anzahl der Tasks ist theoretisch nur vom vorhandenen Speicherplatz abhängig.

Allerdings verlangsamt sich ab einer bestimmten Task-Anzahl die Ausführungszeit unter ein akzeptierbares Maß. Dies hat zwei Gründe: Zum einen muß ein Task lange »anstehen«, bis er weiterbearbeitet wird, zum andern erfordert die Verwaltung und insbesondere das Umschalten der Tasks (das sogenannte »Task-Switching«) ebenfalls Rechenzeit.

Da man die Anzahl der Tasks zum Zeitpunkt des Programmlaufs niemals vorausbestimmen kann, ist zum Beispiel der Versuch, mittels einer Warteschleife die Programmausführung um eine definierte Zeit zu verzögern, von vornherein zum Scheitern verurteilt!

An dieser Stelle setzt man die »Delay()«- oder eine der »Wait()«-Funktionen ein (siehe unten). Diese Funktionen versetzen den Task in einen Wartezustand. Tasks, die sich in diesem Zustand (»Taskstate« genannt) befinden, arbeitet der Prozessor so lange nicht ab, bis sie einen neuen Zustand (»ready«) erhalten. Dies geschieht erst wieder nach dem Eintreten des erwarteten Ereignisses. Erst dann erhebt ihn Exec wieder in die Reihe der auszuführenden Tasks. Nur der Task, den der Prozessor gerade bearbeitet, ist im »running« State. Das heißt für uns, Warteschleifen in dieser Form:

```
for(i=0;i<100000;i++);
```

wechseln ständig zwischen den Zuständen ready und running, nutzen die Möglichkeit des Zustands waiting nicht. Sie belasten daher unnötig das gesamte System, denn alle anderen Tasks müssen ja länger auf ihren nächsten running state warten.

Machen Sie doch einmal den Versuch und testen Sie ein Listing mit Warteschleife, aber ohne die Wait()-Funktion. Starten Sie das Programm und rufen Sie dann einige CLI-Kommandos auf. Sie werden feststellen, daß jetzt deren Ausführgeschwindigkeit merklich nachläßt.

Belastung durch Schleifen

Neben den drei Hauptzuständen waiting, ready und running kann sich ein Task noch in drei Übergangszuständen befinden: Ein Task, der gerade in die »Warteschlange« eingereiht wird, ist vom State »added«, abgelaufene, aber gerade noch vorhandene Tasks vom State »removed« und solche, die sich in einem besonderem Zustand befinden, vom Taskstate »exception«.

Exception-Tasks kennen Sie sicher alle, es handelt sich hierbei zum Beispiel um fehlerhafte Tasks, die eine hübsche Fehlermeldung, auch »Guru Meditation« genannt, nach sich ziehen. Doch nun genug der langen Vorrede, wir haben noch viel zu tun.

Wie einfach ist doch die Arbeit für den Amiga-Anwender: Ein Klick mit der Maus genügt, schon öffnet sich ein Fenster auf der Workbench, welches beispielsweise über den Inhalt eines Verzeichnisses informiert. Was in seiner »Kiste« dabei vorgeht, interessiert ihn recht wenig. Für ihn ist die Hauptsache: Es funktioniert.

Fortsetzung auf Seite 30

Die Amiga-Libraries

clist.library	Funktionen zu Stringmanipulationen
console.library	Arbeiten mit der Tastatur
diskfont.library	Schriftarten auf der Workbench
dos.library	Disk Operating System
exec.library	der Kern des Betriebssystems
expansion.library	für Erweiterungen
graphics.library	Grafik, Copper, Blitter
icon.library	Verwaltung der Icons
info.library	Info Über File, Verzeichnis, Disk
intuition.library	Benutzeroberfläche, Maus
janus.library	Verwendung des SideCar
layers.library	Arbeiten mit Layers (Grafikelement)
mathffp.library	Fast-Floating-Point Mathematik-bibliothek
mathieeedoubbas.library	Erweiterung von mathffp auf IEEE Standard
mathtrans.library	Weitere Mathematikfunktionen
potgo.library	Kontrolle von Joystick, Lightpen etc.
timer.library	Ansprechen der Software-Uhr
translator.library	englische Wörter in Phoneme wandeln

Tabelle 2. Die wichtigsten Libraries des Betriebssystems und deren Aufgabenbereich

Programmname:	Tool.h
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec 3.4 oder 3.6
Aufrufe:	Einbinden mit #include

```
Programmautor: Arno Gölzer
--------------------------

#include <intuition/intuitionbase.h>
#include <libraries/dos.h>
#include <graphics/gfxmacros.h>
#include <graphics/sprite.h>
#include <devices/printer.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <functions.h>
#define ACTIVE_SCREEN ((struct Screen *)(IntuitionBase?Int
uitionBase->ActiveScreen:NULL))
#define ACTIVE_WINDOW ((struct Window *)(IntuitionBase?Int
uitionBase->ActiveWindow:NULL))
#define CircFill(RP,col,x,y,r) EllFill(RP,col,x,y,r,r)
#define FreeFile(buffer,size) FreeMem(buffer,size)
#define TYPE (eintrag->fib_DirEntryType)
#define NAME (eintrag->fib_FileName)
#define SIZE (eintrag->fib_Size)
#define REQ   0
#define DMREQ 1
#define DIRS  2
#define FILES 4
#define FLEN 31
struct IntuitionBase *IntuitionBase=NULL;
struct GfxBase       *GfxBase=NULL;
/*
\\   Libraries öffnen/schließen
*/
SHORT OpenLib()
{/* Libraries öffnen */
    if(!(IntuitionBase=(struct IntuitionBase *)
       OpenLibrary("intuition.library",0)))   return(1);
    if(!(GfxBase=(struct GfxBase *)
       OpenLibrary("graphics.library",0)))    return(2);
    return(0);
}
VOID CloseLib()
{/* Libraries schließen */
    if(GfxBase)       CloseLibrary(GfxBase);
    if(IntuitionBase) CloseLibrary(IntuitionBase);
}
/*
\\        Intuition-Tools
*/
struct Screen *GetScreen(top,depth,modes,title)
    SHORT top,depth;
    USHORT modes;
    STRPTR title;
{/* Screen öffnen */
    struct NewScreen nscr;
    nscr.LeftEdge     = 0;
    nscr.TopEdge      = top;
    nscr.Width        = 320*((modes&HIRES)?2:1);
    nscr.Height       = 256*((modes&LACE) ?2:1);
    nscr.Depth        = depth;
    nscr.DetailPen    = 0;
    nscr.BlockPen     = 1;
    nscr.ViewModes    = modes;
    nscr.Type         = WBENCHSCREEN;
    nscr.Font         = NULL;
    nscr.DefaultTitle = title;
    nscr.Gadgets      = NULL;
    nscr.CustomBitMap = NULL;
    return(OpenScreen(&nscr));
}
struct Window *GetWindow(scr,left,top,width,height,idcmp,fl
ags,title,gad)
    struct Screen *scr;
    SHORT left,top,width,height;
    ULONG idcmp,flags;
    STRPTR title;
    struct Gadget *gad;
{/* Window öffnen */
    struct NewWindow nwin;
    scr=scr?scr:ACTIVE_SCREEN;
    nwin.LeftEdge    = left;
    nwin.TopEdge     = top;
    nwin.Width       = width;
    nwin.Height      = height;
    nwin.DetailPen   = -1;
    nwin.BlockPen    = -1;
    nwin.IDCMPFlags  = idcmp;
    nwin.Flags       = flags;
    nwin.FirstGadget= gad;
    nwin.CheckMark   = NULL;
    nwin.Title       = title;
    nwin.Screen      = scr;
    nwin.BitMap      = NULL;
    nwin.MinWidth    = 20;
    nwin.MinHeight   = 20;
    nwin.MaxWidth    = (scr->Width)-left;
    nwin.MaxHeight   = (scr->Height)-top;
    nwin.Type        = WBENCHSCREEN;
    return(OpenWindow(&nwin));
}
SHORT GetGadget(gad,WorG,left,top,width,height,f,act,type,g
r,sr,text,info,id)
    struct Gadget *gad;
    APTR WorG; /* struct Window oder struct Gadget */
    struct IntuiText *text;
    APTR gr,sr,info;
    SHORT left,top,width,height,id;
    USHORT f,act,type;
{/* Gadget-Struktur initialisieren und in Gadget-Liste einf
gen */
    SHORT nr;
    gad->NextGadget    = (type&REQGADGET)?(struct Gadget *)
    WorG:NULL;
    gad->LeftEdge      = left;
    gad->TopEdge       = top;
    gad->Width         = width;
    gad->Height        = height;
    gad->Flags         = f;
    gad->Activation    = act;
    gad->GadgetType    = type;
    gad->GadgetRender = gr;
    gad->SelectRender = sr;
    gad->GadgetText    = text;
    gad->MutualExclude= NULL;
    gad->SpecialInfo   = info;
    gad->GadgetID      = id;
    gad->UserData      = NULL;
    if(type&REQGADGET)    return(0);
    nr=AddGadget(WorG,gad,-1);
    RefreshGadgets(gad,WorG,NULL);
    return(nr);
}
VOID GetMenu(men,win,left,width,flags,titel)
    struct Menu   *men;
    struct Window *win;
    SHORT left,width;
    USHORT flags;
    BYTE *titel;
{/* Menu-Struktur initialisieren und in die Menu-Liste einf
gen */
    men->NextMenu  = win->MenuStrip;
    men->LeftEdge  = left;
    men->TopEdge   = 0;
    men->Width     = width;
    men->Height    = 10;
    men->Flags     = flags;
    men->MenuName  = titel;
    men->FirstItem = NULL;
    win->MenuStrip = men;
}
VOID GetItem(menu,item,left,top,width,height,flags,mutex,If
ill,Sfill,com)
    struct Menu *menu;
    struct MenuItem *item;
    SHORT left,top,width,height;
    USHORT flags;
    LONG mutex;
    APTR Ifill,Sfill;
    BYTE com;
{/* MenuItem-Struktur initialisieren und in die Item-Liste
einfügen */
    item->NextItem    = menu->FirstItem;
    item->LeftEdge    = left;
```

```
Tg      item->TopEdge      = top;
0Q      item->Width        = width;
uG      item->Height       = height;
U0      item->Flags        = flags;
J5      item->MutualExclude= mutex;
J6      item->ItemFill     = Ifill;
hV      item->SelectFill   = Sfill;
VX      item->Command      = com;
E7      item->SubItem      = NULL;
Sb      menu->FirstItem    = item;
Z40 }
4k  VOID GetSItem(item,sitem,left,top,width,height;flags,mutex,
    Ifill,Sfill,com)
Iy4     struct MenuItem *item;
QE      struct MenuItem *sitem;
3W      SHORT left,top,width,height;
hT      USHORT flags;
R7      LONG mutex;
j3      APTR Ifill,Sfill;
1x      BYTE com;
1w0 {/* SubItem initialisieren und in die SubItem-Liste einfüge
    n */
KD4     sitem->NextItem     = item->SubItem;
vl      sitem->LeftEdge     = left;
ih      sitem->TopEdge      = top;
Sy      sitem->Width        = width;
Iq      sitem->Height       = height;
Qo      sitem->Flags        = flags;
71      sitem->MutualExclude= mutex;
8C      sitem->ItemFill     = Ifill;
Xm      sitem->SelectFill   = Sfill;
Z5      sitem->Command      = com;
9X      sitem->SubItem      = NULL;
Tg      item->SubItem       = sitem;
vQ0 }
jF  SHORT GetRequest(req,win,l,t,w,h,rl,rt,gad,rand,text,flags,
    bf,dm)
3z4     struct Requester *req;
B2      struct Window *win;
uL      SHORT l,t,w,h,rl,rt,dm;
Rq      struct Gadget *gad;
J8      struct Border *rand;
Li      struct IntuiText *text;
6s      USHORT flags;
1A      UBYTE bf;
IM0 {/* frei definierbares Requester installieren */
Zy4     InitRequester(req);
Yd      req->LeftEdge =l;
ym      req->TopEdge  =t;
fW      req->Width    =w;
V2      req->Height   =h;
cd      req->RelLeft  =rl;
tK      req->RelTop   =rt;
et      req->ReqGadget=gad;
gL      req->ReqBorder=rand;
HJ      req->ReqText  =text;
qI      req->Flags    =flags;
jZ      req->BackFill =bf;
sF      if(dm==DMREQ) return(SetDMRequest(win,req));
fa      return(Request(req,win));
Kp0 }
3G  ULONG GetMessage(win,code,id,qual)
ZQ4     struct Window *win;
0g      USHORT *code,*id,*qual;
510 {/* MsgPort auslesen; liefert CLASS,CODE,QUALIFIER der Msg
    und GadgetID */
2m4     struct IntuiMessage *msg;
UX      ULONG class=0;
A8      if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
2T8         class=msg->Class;
9H          if(code) *code=msg->Code;
d3          if(id)   *id  =(((struct Gadget *)msg->IAddress)-
            >GadgetID);
H7          if(qual) *qual=msg->Qualifier;
0W          ReplyMsg(msg);
X24     }
B6      else{
W28         if(code) *code=0;
qf          if(id)   *id  =0;
cQ          if(qual) *qual=0;
c74     }
vY      return(class);
e90 }
d2  /*
cn  \\          Grafik-Tools
k9  */
81  VOID Line(RP, col, x1, y1, x2, y2)
Nb4     struct RastPort *RP;
J6      SHORT col, x1, y1, x2, y2;
OY0 {/* Linie zeichnen */
n04     SetDrMd(RP, JAM1);
jt      SetAPen(RP, col);
Yw      Move(RP,x1,y1);
gs      Draw(RP,x2,y2);
qL0 }
16  VOID ILine(RP, x1, y1, x2, y2)
Wk4     struct RastPort *RP;
po      SHORT x1, y1, x2, y2;
vX0 {/* Linie im JAM1|COMPLEMENT-Modus zeichnen */
e94     SetDrMd(RP, JAM1|COMPLEMENT);
g4      Move(RP,x1,y1);
o0      Draw(RP,x2,y2);
yT0 }
9C  VOID Block(RP, col, x1, y1, x2, y2)
es4     struct RastPort *RP;
re      SHORT col,x1,y1,x2,y2;
wo0 {/* Block zeichnen */
Jf4     if(!((x1>x2) || (y1>y2))){
5g8         SetDrMd(RP, JAM1);
1B          SetAPen(RP, col);
fj          RectFill(RP, x1, y1, x2, y2);
7c4     }
8d0 }
Bb  VOID IBlock(RP, x1, y1, x2, y2)
o24     struct RastPort *RP;
h5      SHORT x1,y1,x2,y2;
Uo0 {/* Block im JAM1|COMPLEMENT-Modus zeichnen */
Tp4     if(!((x1>x2) || (y1>y2))){
xS8         SetDrMd(RP, JAM1|COMPLEMENT);
os          RectFill(RP, x1, y1, x2, y2);
G14     }
Hm0 }
7H  VOID IRast(RP)
xB4     struct RastPort *RP;
U40 {/* RastPort im JAM1|COMPLEMENT-Modus darstellen */
W44     IBlock(RP,0,0,RP->BitMap->BytesPerRow*8,RP->BitMap-
        >Rows);
Mr0 }
jf  VOID GetPos(RP, x, y)
2G4     struct RastPort *RP;
a4      SHORT *x, *y;
hf0 {/* Position des Grafikcursors im RastPort ermitteln */
S44     *x=RP->cp_x;
YC      *y=RP->cp_y;
Ty0 }
Hu  SHORT GetCol(VP,nr, r,g,b)
3U4     struct ViewPort *VP;
Hq      SHORT nr, *r,*g,*b;
C20 {/* Farbregister auslesen */
Jp4     SHORT col;
FT      col= GetRGB4(VP->ColorMap,nr);
fB      *r = (col>>8)&15;
xE      *g = (col>>4)&15;
TW      *b = col&15;
Nf      return(col);
e90 }
4A  VOID Print(RP, text, col, xpos, ypos)
KY4     struct RastPort *RP;
70      STRPTR text;
31      SHORT col, xpos, ypos;
Wk0 {/* Ausgabe eines Textes */
1M4     SetDrMd(RP, JAM1);
hr      SetAPen(RP, col);
5L      Move(RP, xpos, ypos+RP->TxBaseline);
ta      Text(RP, text, strlen(text));
oJ0 }
2T  SHORT EllFill(RP, col, x, y, rh, rv)
Ui4     struct RastPort *RP;
9Y      SHORT col, x, y, rh, rv;
nE0 {/* ausgefüllte Ellipse in beliebiger Farbe zeichnen */
sc4     struct TmpRas *Raster;
65      struct AreaInfo *AInfo;
CS      PLANEPTR Bitplane;
pt      WORD Buffer[10];
```

Listing 1. Tool.h – unser Headerfile erleichtert Ihnen den Einstieg in die Systemprogrammierung des Amiga.

```
    BOOL error=1;
    SHORT height=3*rh,width=3*rv;
    if((Bitplane=AllocRaster(width,height))&&
       (Raster=(struct TmpRas *)AllocMem(sizeof(struct TmpR
       as),MEMF_CLEAR))&&
       (AInfo=(struct AreaInfo *)AllocMem(sizeof(struct Are
       aInfo),MEMF_CLEAR))){
          SetAPen(RP,col);
          InitTmpRas(Raster,Bitplane,RASSIZE(width,height)
          );
          RP->TmpRas=Raster;
          InitArea(AInfo,Buffer,2);
          RP->AreaInfo=AInfo;
          error=AreaEllipse(RP,x,y,rh,rv);
          AreaEnd(RP);
    }
    if(AInfo)    FreeMem(AInfo,sizeof(struct AreaInfo));
    if(Raster)   FreeMem(Raster,sizeof(struct TmpRas));
    if(Bitplane) FreeRaster(Bitplane,width,height);
    return(error);
}
BOOL HrdCpy(scr,left,top,width,height,size)
    struct Screen *scr;
    UWORD left,top,width,height;
    USHORT size;
{/* Screen-RastPort ausdrucken */
    union PrIO{
        struct IOStdReq ios;
        struct IODRPReq iodrp;
        struct IOPrtCmdReq iopc;
    }*request,*CreateExtIO();
    struct MsgPort *MPort;
    BOOL OK=1;
    scr=scr?scr:ACTIVE_SCREEN;
    if(!(MPort=CreatePort("TOOL_Port",0))) return(1);
    if(request=(union PrIO *)CreateExtIO(MPort,sizeof(union
     PrIO))){
        if(!(OpenDevice("printer.device",0,request,0))){
            request->iodrp.io_Command=PRD_DUMPRPORT;
            request->iodrp.io_RastPort=&scr->RastPort;
            request->iodrp.io_ColorMap=scr->ViewPort.Colo
            rMap;
            request->iodrp.io_Modes=scr->ViewPort.Modes;
            request->iodrp.io_SrcX=left;
            request->iodrp.io_SrcY=top;
            request->iodrp.io_SrcWidth=width;
            request->iodrp.io_SrcHeight=height;
            request->iodrp.io_DestCols=(LONG)(size*width);
            request->iodrp.io_DestRows=(LONG)(size*height)
            ;
            request->iodrp.io_Special=SPECIAL_MILCOLS|
                                      SPECIAL_MILROWS|
                                      SPECIAL_ASPECT;
            DoIO(request);
            CloseDevice(request);
        }
        else OK=0;
        DeleteExtIO(request,sizeof(union PrIO));
    }
    else OK=0;
    DeletePort(MPort);
    return(OK);
}
/*
\\          DOS-Tools
*/
LONG Exists(name)
    STRPTR name;
{/* Existenz eines Eintrags feststellen */
    LONG bytes=0;
    struct FileInfoBlock *eintrag;
    struct FileLock *lock;
    if(lock=(struct FileLock *)Lock(&name[0],ACCESS_READ)){
        if(eintrag=(struct FileInfoBlock *)
            AllocMem(sizeof(struct FileInfoBlock),MEMF_CLEA
            R)){
            if(Examine(lock,eintrag)) bytes=(TYPE>0) ? -1
            : SIZE;
```

```
IU              FreeMem(eintrag,sizeof(struct FileInfoBlock));
6b8         }
SX          UnLock(lock);
8d4     }
oB      return(bytes);
AfO }
Jq  UBYTE *GetFile(file,fsize)
O24     STRPTR file;
Db      LONG *fsize;
qoO {/* File in Speicher laden */
7E4     struct FileHandle *FH;
VM      UBYTE *buffer;
HS      LONG size;
EN      if((size=Exists(file))<=0) return(0);
qJ      if(!(buffer=(UBYTE *)AllocMem(size,MEMF_CLEAR))) return
        (0);
ql      else if(!(FH=(struct FileHandle *)Open(file,MODE_OLDFIL
        E))){
7I8         FreeMem(buffer,size);
pk          return(0);
Ns4     }
KX      else if(Read(FH,buffer,size)!=size){
BM8         FreeMem(buffer,size);
iS          Close(FH);
up          return(0);
Sx4     }
61      else{
mW8         Close(FH);
H7          *fsize=size;
js          return(buffer);
X24     }
Y3O }
Bm  BOOL PutFile(file,buffer,size)
mQ4     STRPTR file;
qh      UBYTE *buffer;
cn      LONG size;
t60 {/* Buffer speichern */
Wd4     struct FileHandle *FH;
Fj      BOOL rw=0;
ht      if(!(FH=(struct FileHandle *)Open(file,MODE_NEWFILE)))
        return(1);
kw      if(Write(FH,buffer,size)!=size)     rw=1;
Ok      Close(FH);
si      return(rw);
kFO }
9W  LONG GetLW()
lf  {/* Nummer des aktuellen Laufwerks ermitteln */
DQ4     struct FileInfoBlock *eintrag;
ij      struct InfoData *infos;
gJ      struct FileLock *lock;
ZH      LONG disk=-1;
Iq      if(infos=(struct InfoData *)AllocMem(sizeof(struct Info
        Data),MEMF_CLEAR)){
Wk8         if(lock=(struct FileLock *)Lock(NULL,ACCESS_READ)){
5mC             if(eintrag=(struct FileInfoBlock *)
JHG                 AllocMem(sizeof(struct FileInfoBlock),MEMF_
                    CLEAR)){
d0                  if(Examine(lock,eintrag)){
92K                     Info(lock,infos);
f4                      disk=infos->id_UnitNumber+1;
yTG                 }
CO                  FreeMem(eintrag,sizeof(struct FileInfoBlock
                    ));
OVC             }
MR              UnLock(lock);
2X8         }
w1          FreeMem(infos,sizeof(struct InfoData));
4Z4     }
53      return(disk);
6bO }
sO  SHORT ReadDir(dirname,anz,flags,namen,size)
3e4     STRPTR dirname;
Hu      USHORT anz;
EV      SHORT flags;
I9      UBYTE namen[][FLEN];
```

Listing 1. (Fortsetzung)

```
        LONG size[];
    {/* Verzeichnis lesen */
        SHORT i=-1;
        struct FileInfoBlock *eintrag;
        struct FileLock *lock;
        if(lock=(struct FileLock *)Lock(&dirname[0],ACCESS_READ
        )){
            if(eintrag=(struct FileInfoBlock *)
                AllocMem(sizeof(struct FileInfoBlock),MEMF_CLEA
                R)){
                if(Examine(lock,eintrag)){
                    if(TYPE>0){
                        for(i=anz;i>0;i--) namen[i-1][0]='\0';
                        ExNext(lock,eintrag);
                        while((IoErr()!=ERROR_NO_MORE_ENTRIES)&
                        &(i<anz)){
                            if(TYPE>0){
                                if(flags&DIRS){
                                    strcpy(&namen[i][0],&NAME[0
                                    ]);
                                    if(size) size[i]=-1;
                                    i++;
                                }
                            }
                            else if(flags&FILES){
                                strcpy(&namen[i][0],&NAME[0]);
                                if(size) size[i]=SIZE;
                                i++;
                            }
                            ExNext(lock,eintrag);
                        }
                    }
                }
                FreeMem(eintrag,sizeof(struct FileInfoBlock));
            }
            UnLock(lock);
        }
        return(i);
    }
(C) 1988 M&T
```

Listing 1. (Schluß)

Fortsetzung von Seite 25

Der Programmierer wird sich schon eher Gedanken machen, denn jede Funktion, die im Computer, genauer im ROM (»ReadOnlyMemory»), integriert ist, muß er nicht formulieren, er kann sie mitbenutzen.

Das ROM des Amiga enthält eine Vielzahl von Routinen, die nur darauf warten, gestartet zu werden. Nur wo findet man diese? Keine Angst, Sie müssen keine ROM-Listings wälzen! Der Amiga stellt uns zu den verschiedenen Betriebssystemteilen sogenannte Libraries zur Verfügung. Diese Libraries sind Sprungtabellen zu den einzelnen Routinen. Um Platz zu sparen, sind nicht alle Bibliotheken im ROM. Diejenigen, die seltener benötigt werden, finden Sie auf der Workbench im Verzeichnis »LIBS«, sie werden bei Bedarf nachgeladen. Tabelle 2 zeigt alle zur Zeit verfügbaren Libraries und deren Aufgabengebiet. Abgesehen von der Tatsache, daß uns das Studium des Amiga-ROM-Listings erspart bleibt, hat die Verwendung von Libraries einen zweiten entscheidenden Vorteil: Bei Überarbeitung des Betriebssystems müssen nicht alle Programme geändert, sondern lediglich die Tabellen in den Libraries aktualisiert werden.

Wenn Sie Ihren Amiga einschalten (oder nach einem Reset), ist nur eine einzige Library ansprechbar, die »exec.library«. Alle anderen werden je nach Bedarf vom Programm aus geöffnet, wie zum Beispiel die »dos.library« vom Startup-Programm.

Systemroutinen mitbenutzen

»Öffnen« bedeutet die Basisadresse der Sprungtabelle ermitteln – die Tabellen befinden sich nämlich nicht immer an der gleichen Stelle im Speicher. Wenn diese Basisadresse bekannt ist, kann über einen sogenannten Offset, das ist die Differenz der Basisadresse zur Adresse des JMP-Befehls in der Tabelle, die Funktion aufgerufen werden.

Da uns direkt nach dem Einschalten beziehungsweise Reset nur die exec.library zur Verfügung steht, muß die Funktion zum Öffnen einer Library in der Exec-Sprungtabelle zu finden sein. Das bedeutet aber, daß auch die Basisadresse der exec.library bekannt sein muß. Das ist korrekt – diese Adresse steht immer ab der Speicherstelle 4, der einzigen Speicherstelle im Amiga, die immer die gleiche Information enthält. Man erhält diese Adresse auch über die vordefinierte Konstante »SysBase« (siehe Tabelle 3).

Aber das Arbeiten mit Offsets überlassen wir besser dem Compiler. Wir sprechen unsere Funktionen einfach mit ihrem Namen an. Etwa so:

```
OpenLibrary("intuition.
library",0);
```

Die Exec-Funktion »OpenLibrary()« öffnet die für Intuition zuständige Bibliothek. Wie Sie sehen, benötigt die Funktion zwei Parameter; als ersten den Library-Namen und als zweiten die Versionsnummer der Library. Null als Versionsnummer fordert die aktuelle (neueste) Library-Version.

»OpenLibrary()« liefert die Adresse der Struktur, die die notwendigen Informationen der gewünschten Library enthält. Entsprechend muß eine Zeigervariable (32 Bit) definiert werden, die diese Adresse aufnehmen kann.

Der Variablenname kann jedoch nicht frei gewählt werden. Tabelle 3 zeigt eine Aufstellung aller fest vereinbarten Variablennamen und der entsprechenden Libraries. Die Zeigervariable muß darüber hinaus global, also außerhalb jeder Funktion vereinbart sein, so daß zum Beispiel das Öffnen der Intuition.library die in Listing 2 definierte Sequenz verlangt. »IntuitionBase« ist seinerseits wieder eine Struktur, die im Headerfile »intuition/intuitionbase.h« definiert ist. Sie enthält als erstes Element eine Library-Struktur, die durch den Aufruf von OpenLibrary() initialisiert wird.

Mittels des »Cast«-Operators greifen wir einer Konvertierung vor. Konnte die Bibliothek nicht geöffnet werden, liefert OpenLibrary() den Wert »NULL«.

Fassen wir zusammen: Bevor wir mit dem Programmieren beginnen können, müssen wir uns überlegen, welchen Teil des Amiga-Betriebssystems wir mitbenutzen möchten und dann die entsprechende Library öffnen. Diese Prozedur muß auch bei dem kleinsten Programm wiederholt werden. Hier findet unser Headerfile seinen ersten Einsatz: »Tool.h« kann

Arno Gölzer



sammelte seine ersten Computer-Erfahrungen – wie zwei Millionen andere Computerfreaks – auf einem C64. Ersten Programmierversuchen in Basic folgte der Umstieg auf das schnellere Assembler.

Die Leistungsfähigkeit dieser Sprache stärkte seine Treue zu jenem phantastischen Computer – allerdings nur bis zum Erscheinen des Amiga. Dieser stand am Anfang einer steilen Programmierer-Karriere:

Arno Gölzer veröffentlichte bereits mehrere Programme im AMIGA-Magazin, zwei davon wurden Listing des Monats, »Volltreffer«, eine Billardsimulation, Listing des Jahres. Daneben veröffentlichte er – ebenfalls im AMIGA-Magazin – C-Kurse für Anfänger und Fortgeschrittene. In diesem Sonderheft finden Sie neben dem neuen C-Kurs, der Ihnen das Amiga-Betriebssystem nahebringt, eine Shell, ein Ray-Tracing-Programm und einen Sound-Editor aus seiner »Feder«.

Arno Gölzer begann nach der mittleren Reife eine Lehre in Elektronik und Maschinenbau. Seit 1987 studiert er Maschinenbautechnik.

```
struct IntuitionBase *IntuitionBase;
....
funktion()
{
....
IntuitionBase=(struct IntuitionBase *)OpenLibrary("intuition.
library",0);
if(IntuitionBase==NULL) exit(ERROR);
....
}
```

Listing 2. Ein Beispiel für das Öffnen einer Library

uns diese Arbeit abnehmen. Schauen wir uns das Listing etwas genauer an.

Zu Beginn werden alle notwendigen Headerfiles eingebunden. Da die Eingliederung von Headerfiles in Headerfiles erlaubt ist, können wir alle benötigten von Tool.h aus einbinden und uns so lästige Schreibarbeit sparen. Ein sehr wichtiges und interessantes Headerfile ist »functions.h«. Dort werden fast alle Betriebssystemroutinen deklariert. Mit der Einbindung ersparen wir uns lästige »Warnings«. Für den Lattice-Compiler finden Sie vergleichbare Haeder-Files unter den sogenannten »Prototypes«. Auch dieses File sollten Sie sich genau ansehen. Im weiteren Verlauf dieses Kurses verweisen wir immer wieder auf interessante Headerfiles. Machen Sie sich bitte die Mühe, diese aufmerksam durchzuarbeiten. Es werden Ihnen dann sicher Zusammenhänge klar, die Ihnen sonst verborgen bleiben.

Im nächsten Tool.h-Abschnitt finden Sie einige Symboldefinitionen und Makros. Doch darüber werden Sie mehr erfahren, wenn diese benutzt werden. Es folgt die globale Definition der »Base«-Adressen, die oben erwähnt wurden. Die Variable IntuitionBase wurde zum Beispiel in diesem Bereich als Zeiger auf die IntuitionBase-Struktur vereinbart. Die Funktionen schließen sich an. Sie sind untergliedert in Libraries, Intuition-, Grafik- und DOS-Tools. Beginnen wir mit der gleich am Anfang definierten Funktion OpenLib(). Sie besorgt das Öffnen der notwendigen Libraries. In der vorliegenden Tool.h-Version sind dies die Intuition.library und die Graphics.library. Die letztere beinhaltet, wie ihr Name schon andeutet, die Adressen der Grafikroutinen. Später können Sie Ihr Tool natürlich erweitern.

```
/*
\ \ Beep.c
*/
#include "TOOL.h"
VOID main() /* Hauptfunktion */
{
if(OpenLib()){
printf("Kann Intuition nicht öffnen!\ n");
}
else{
DisplayBeep(0);
CloseLib();
}
}
```

Listing 3. Der erste Einsatz von »Tool.h« – ein Bildschirmblitz

OpenLib() kehrt mit dem Wert 0 zurück, wenn alle gewünschten Libraries geöffnet sind. Im anderen Fall entspricht der Return-Wert der Nummer der Bibliothek, welche OpenLib() nicht öffnen konnte. Das heißt also, wenn sich Intuition nicht öffnen läßt, ist das Ergebnis von OpenLib() eins, wenn sich die GraphicsLibrary nicht öffnen läßt zwei und so weiter.

CloseLib(), die nächste Funktion in unserem Headerfile, ist – wie der Name vermuten läßt – das Gegenstück zu OpenLib(). Sie schließt alle bisher geöffneten Libraries.

Im nächsten großen Abschnitt sind die Intuition-Tools aufgelistet, die Routinen zu dem Betriebssystemteil, welchen wir als erstes beleuchten wollen.

Intuition, was verbirgt sich hinter diesem Begriff? Im Zusammenhang mit dem Amiga hört man sehr oft davon.

Intuition? Intuition!

Im Wörterbuch findet man hierfür »Eingebung« oder »verwickelte Zusammenhänge sofort richtig erfassen«, und das trifft den Nagel auf den Kopf. Gemeint ist nämlich die Benutzeroberfläche des Amiga mit den Screens, den Windows, Menüs und Gadgets. Hier findet man sich auch als Anfänger rasch zurecht – man erfaßt sofort die Zusammenhänge.

Bevor wir nun die Intuition-Tools im einzelnen besprechen, wagen wir doch einen kleinen Versuch mit den bereits besprochenen Funktionen. »Beep.c« (Listing 3) läßt alle zur Zeit des Aufrufs geöffneten Screens aufblitzen. Das Programm macht sich gut als CLI-Kommando, etwa um den Anwender auf einen Fehler während der Ausführung einer Batchdatei, zum Beispiel der Startup-Sequence, aufmerksam zu machen.

Schauen wir uns das Programm gemeinsam an: In der ersten Zeile wird unser Headerfile mit Hilfe der »#include«-Anweisung eingebunden. Diese Anweisung veranlaßt den Preprozessor, die Zeile vor der Compilierung durch das angegebene File zu ersetzen. Sonst ist in unserem kleinen Programm nur noch die Hauptfunktion »main()« definiert. Dort rufen wir zu Beginn die Funktion OpenLib() auf. Sollte sie einen Wert ungleich »NULL« liefern, erfolgt nach der Ausgabe einer entsprechenden Fehlermeldung sofort das Programmende.

Sind die Libraries erfolgreich geöffnet, starten wir auch gleich unsere erste Intuition-Systemroutine: »DisplayBeep()«. Diese Funktion läßt wahlweise einen oder auch alle geöffneten Screens (siehe unten) aufblitzen. Als Parameter benötigt sie einen Zeiger auf den Screen, der blitzen soll. Wenn Sie einen Nullzeiger übergeben, ist der Blitz auf jedem Screen zu sehen.

```
DisplayBeep(&Screen);  /
* der Bildschirm mit der
Adresse  »Screen«
blitzt */
DisplayBeep(0);        /
* alle geöffneten Bild-
schirme (Screens)
blitzen */
```

Kinderleicht, oder? Als letzten Schritt schließen wir mit Hilfe von »CloseLib()« alle Libraries. Das ist sehr wichtig, denn die Libraries werden beim Programmende nicht automatisch geschlossen. Genauso verhält es sich beim Öffnen von Screens, Windows, Devices oder beim Reservieren von Speicherplätzen, aber davon später mehr. Zuerst müssen wir einmal ergründen, was wir da gerade zum Blitzen gebracht haben.



...ist die Basis-Darstellungsfläche für alle Intuition-Elemente. Ohne Screen kein Fenster, keine Menüs, noch nicht einmal das kleinste Gadget. Das beste Beispiel hierfür ist der Workbenchscreen. Geben Sie doch einmal aus dem CLI heraus das Kommando »ENDCLI« ein, ohne vorher mit »LOADWB« die Workbench gestartet zu haben. Der Workbenchscreen bleibt geschlossen und es wird etwas langweilig.

Der Amiga ist in der Lage, eine beliebige Anzahl verschiedener Bildschirme gleichzeitig zu verwalten. Allein der Speicher setzt uns hier Grenzen, denn die Daten der Screens liegen im sogenannten Chip-Memory, das sind die untersten 512 KByte im Speicher. Diese Daten sind übrigens nicht fest an Adressen gebunden, wie zum Beispiel der Bildschirmspeicher des C64.

Durch die Vereinbarung verschiedener Parameter wird das Aussehen eines Screens, (Größe, Position, Auflösung und so weiter) bestimmt. Diese Parameter werden in einer besonderen Struktur, der NewScreen-Struktur übergeben. Diese Struktur (Listing 4) ist im Headerfile »intuition.h« definiert. Hier die Bedeutung der einzel-

Library-Namen

clist	ClistBase
diskfont	DiskfontBase
dos	DosBase
exec	SysBase
expansion	ExpansionBase
graphics	GfxBase
icon	IconBase
intuition	IntuitionBase
layers	LayersBase
mathffp	MathBase
mathieeedoubbas	MathIeeeDoubBasBase
mathtrans	MathTransBase
potgo	PotgoBase
translator	TranslatorBase

Tabelle 3. Der Name der Variablen, die die Basisadresse einer Library enthält, kann nicht beliebig gewählt werden

```
struct NewScreen
{
SHORT LeftEdge;
SHORT TopEdge;
SHORT Width;
SHORT Height
SHORT Depth;
UBYTE DetailPen, BlockPen;
USHORT ViewModes;
USHORT Type;
struct TextAttr *Font;
UBYTE *DefaultTitle;
struct Gadget *Gadgets;
struct BitMap *CustomBitMap;
} ;
```

Listing 4. Die NewScreen-Struktur vereint alle Parameter eines Screens

nen Komponenten dieser wichtigen Struktur:

LeftEdge beschreibt die X-Position der linken Bildschirmseite. Diese Variable hat in der aktuellen Betriebssystemversion noch keine Bedeutung, setzen Sie diese daher stets auf Null.

TopEdge ist die Y-Position der oberen Bildschirmkante, nach dem Öffnen des Screens. Wenn Sie einen größeren Wert als 0 eingeben, wird (wenn mehrere vorhanden sind) der Screen im Hintergrund geöffnet.

Die beiden beschriebenen Variablen geben die Position des Screens an. Screens können in der aktuellen Version des Betriebssystems nach dem Öffnen nur in Y-Richtung verschoben werden. Dazu muß die Titelleiste - das ist der Balken am oberen Screenende - mit der Maus angeklickt und nach unten oder oben geschoben werden.

Width beschreibt die Breite des Bildschirms. Auch wenn Sie hier einen kleineren Wert als die mögliche Auflösung in X-Richtung angeben, wird ein eventuell im Hintergrund liegender Screen nicht sichtbar. Die maximale Breite eines Bildschirms ist abhängig vom eingestellten View-Modus (siehe unten) und beträgt, wenn »HIRES« eingestellt ist 640, ansonsten 320 Punkte.

Height ist die Höhe des Screens. Auch hier bleibt der Hintergrund-Screen verborgen, selbst wenn er, durch die Angabe eines entsprechend kleinen Wertes für die Bildschirmhöhe, theoretisch sichtbar sein könnte. Auch die maximale Höhe ist von der gewählten Auflösung abhängig. Im »LACE«-Modus (siehe unten) beträgt sie 512, im anderen Fall 256 Punkte.

Depth (Tiefe) gibt die Anzahl der Bit-Ebenen an. Die Bilddaten für den Screen sind in verschiedene Ebenen aufgeteilt. Im einfachsten Fall bedeutet ein gesetztes Bit im Speicher einen gesetzten Punkt auf dem Bildschirm. Greifen wir uns einmal einen beliebigen Punkt des Bildschirms heraus. Irgendwo im Speicher ist für diesen Punkt ein Bit entweder gesetzt oder auch nicht gesetzt. Wir erhalten jedenfalls einen Wert für den Zustand dieses Bits, nämlich 0 oder 1. Dieser Wert gibt an, welches Farbregister für den Punkt auf dem Bildschirm zuständig ist. Der Amiga verfügt über 32 solcher Farbregister. Jedes dieser Register kann einen 12 Bit breiten Farbwert aufnehmen, das sind immerhin 4096 Möglichkeiten. Die untersten 4 Bits eines jeden Farbregisters beschreiben den Blau-Anteil, die nächsten 4 den Grün-Anteil und wieder die nächsten 4 den Rot-Anteil der gewünschten Farbe. Da die Farbregister jedoch 16 Bit breit sind, bleiben die 4 obersten Bit (bisher) noch ungenutzt.

Rot, Grün und Blau getrennt

Angenommen, Register 0 ist für die Hintergrundfarbe zuständig und enthält den hexadezimalen Wert »0x0F00« (blau), Register 1 den Wert »0x0FFF« für weiß. Demzufolge muß unser Punkt weiß sein, wenn das entsprechende Bit im Speicher gesetzt ist, denn ein gesetztes Bit entspricht dem Wert 1. Der Punkt erhält daher die Farbe aus Farbregister eins und dieses enthält den Wert für weiß.

Ist das Bit im Speicher nicht gesetzt, so entspricht das dem Wert 0. Die Farbe des Punktes auf dem Screen richtet sich also nach dem Inhalt des Farbregisters 0 - er wird blau dargestellt.

Wie oben erwähnt, hat der Amiga aber 32 Farbregister.

Um diese Palette auszuschöpfen, müssen wir unserem Punkt nur genügend Bits zur Verfügung stellen. Um 32 (= 2 hoch 5) verschiedene Zustände darstellen zu können, benötigen wir 5 Bit. Da wir natürlich die anderen Punkte nicht benachteiligen wollen, erhält jeder seine 5 Bits und so entstehen fünf Bit-Ebenen (Bild 1).

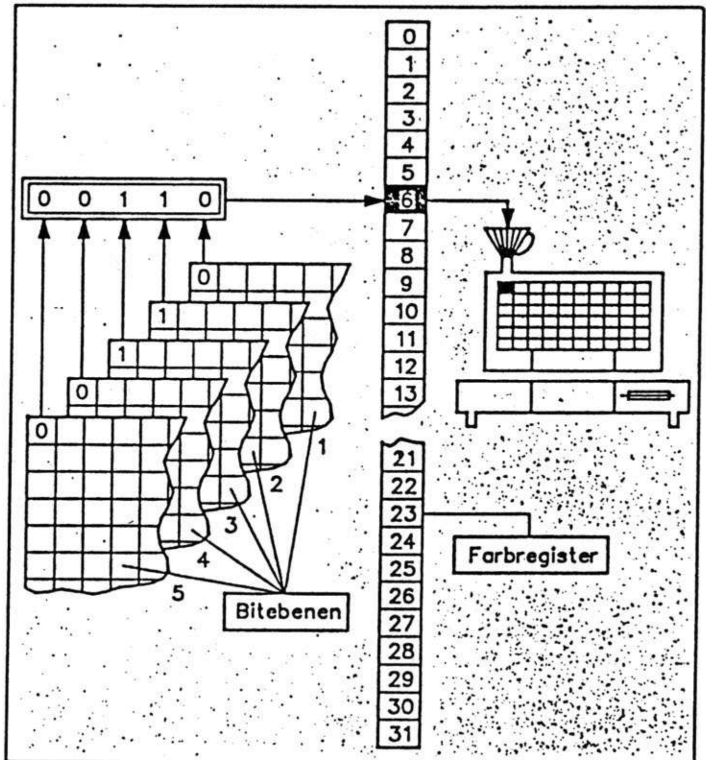


Bild 1. Die Bit-Ebenen legen die Farbnummern eines Punktes auf dem Bildschirm fest

Der benötigte Speicherplatz eines Bildschirms wächst proportional zu der Anzahl der gewünschten Punkte (Auflösung) und ebenfalls proportional zu der Anzahl der Bit-Ebenen. Daher berechnet sich zum Beispiel der Speicherplatz für einen »HIRESLACE«-Bildschirm mit vier Bit-Ebenen, das sind 2^4 = 16 Farben, wie folgt:

```
Memory=(640*512)Punkte*4
Bit-Ebenen
```

Das sind 1310720 Bit oder 163840 Byte. In Tabelle 4 finden Sie die Anzahl der Farben, die zugehörenden Farbregister und den benötigten Speicherplatz für jeden Depth-Wert.

Entsprechend der Anzahl der Farbregister kann die Anzahl der Bitplanes den Wert 5 nicht überschreiten. Dennoch ist eine sechste Bit-Ebene vorhanden.

Diese findet unter anderem in zwei besonderen Modi, dem »EXTRA_HALF_BRITE«- und dem »HAM«-Modus, Verwendung. Diese sind weiter unten ausführlicher beschrieben. Im Moment ist nur wichtig festzuhalten, daß der Wert für Depth zwischen 0 und 7 liegen muß.

DetailPen verwenden Sie, um die Nummer des Farbregisters einzutragen, das die Farbe enthält, mit der die Schrift und die Gadgets der Screen-Titelleiste erscheinen sollen. Oft wird hier der Wert 0 eingetragen.

BlockPen ist die »Stiftfarbe« mit der die Titelleiste des Screens gefüllt wird. Auch hier wird wieder die Nummer des Farbregisters verlangt, das die gewünschte Farbe enthält. Der Wert 1 hat sich hier eingebürgert, natürlich ist auch jeder andere Wert von 0 bis 2^{Depth}-1 erlaubt.

ViewModes legt die Darstellungsmodi des Amiga fest. Durch Setzen verschiedener

Farben u. Speicherplatz f. verschiedene Depth-Werte

Bitplanes	Farben	Farbregister	320*256	320*512	640*256	640*512 Punkte
1	2	0 und 1	10240	20480	20480	40960 Bytes
2	4	0 bis 3	20480	40960	40960	81920 Bytes
3	8	0 bis 7	30720	61440	61440	122880 Bytes
4	16	0 bis 15	40960	81920	81920	163840 Bytes
5	32	0 bis 31	51200	102400	—	—

Tabelle 4. Mit dem Wert für die Depth-Variable der NewScreen-Struktur ändert sich die Anzahl der Farben, aber auch der benötigte Speicherplatz

AMIGA PROGRAMMSERVICE

Sie suchen hilfreiche Utilities und professionelle Anwendungen für Ihren Computer? Sie wünschen sich gute Software zu vernünftigen Preisen? Hier finden Sie beides!
Unser stetig wachsendes Sortiment enthält interessante Listing-Software für alle gängigen Computertypen. Jeden Monat erweitert sich unser aktuelles Angebot um eine weitere interessante Programmsammlung für jeweils einen Computertyp.
Bei Fragen zu Bestellung und Versand der Programmservice-Disketten wählen Sie bitte:
Telefon (089) 4613-232.

Bestellungen bitte *nur gegen Vorauskasse* an: Markt&Technik Verlag AG, Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, D-8013 Haar, Telefon (089) 4613-0.
Schweiz: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656.
Österreich: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0, Microcomput-ique, E. Schiller, Fasangasse 24, A-1030 Wien, Telefon (0222) 785661; Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Telefon (0222) 833196. Ueberreuter Media, Verlagsges. mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.
Bestellungen aus anderen Ländern bitte nur schriftlich an: Markt&Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, D-8013 Haar. Nur gegen Bezahlung der Rechnung im voraus.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die beigeheftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

Amiga Sonderheft 3: Basic, Spiele

Broker: Erleben Sie die Faszination der Börse hautnah. Diese Simulation für 2 bis 4 Spieler ist einzigartig. Der Autor setzt seine fundierten Kenntnisse in spannendes Spielgeschehen um.
Ping-Pong: Dieses Sportspiel bringt Wettkampfstimmung ins Wohnzimmer. Dreidimensionale Darstellung, realistische Soundeffekte und rasante Ballwechsel führen zu lang anhaltendem Spielspaß.
Anpfiff: Als Manager in der Fußball-Bundesliga führen Sie Ihr Lieblingsteam durch die Saison. Zusätzliche Spiele im UEFA-Cup verhelfen Ihrer Mannschaft zu Ruhm und Ihnen zu vielen Manager-Punkten.
Basic-Routinen: Die Basic-Kurse im Sonderheft 3 bieten zahlreiche, hilfreiche Routinen. Alle dort vorgestellten Programme finden Sie auch auf dieser Programmservice-Diskette.
Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des Amiga-Sonderhefts 3 mit einem Diskettensymbol gekennzeichnet sind.
3 1/2"-Diskette für Amiga

Bestell-Nr. 45903 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga Sonderheft 2: Grafik, Anwendung

Object-Editor: Animierte Figuren, beispielsweise für eigene Spiele, entwickeln Sie mit diesem Editor auf komfortable Weise. Sogar mit Deluxe Paint erstellte Pinsel lassen sich einlesen.
Haushaltsbuch: Mit diesem hervorragenden Anwendungsprogramm verwalten Sie alle Einnahmen und Ausgaben auf übersichtliche Weise. Eine Monats- oder Jahresstatistik zeigt, in welchen Bereichen Sie zukünftig sparen können. Jetzt haben Sie Ihre Finanzen im Griff.
Keyboard-Master: Lernen Sie im Zehn-Finger-System zu tippen. Mit diesem didaktisch ausgereiften Programm ist dies kein Problem. Für Programmierer sind sogar Spezial-Lektionen mit wichtigen Sonderzeichen vorhanden.
FastLoadCopy: Dieses Tool bringt den DIR-Befehl auf Trab. Nach der »Operation« wird das Inhaltsverzeichnis einer Diskette im D-Zug-Tempo eingelesen. Zusätzlich kopiert das Programm Disketten und versieht diese mit dem schnellen Directory.
Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des Amiga-Sonderhefts 2 mit einem Diskettensymbol gekennzeichnet sind.
3 1/2"-Diskette für Amiga

Bestell-Nr. 45802 **DM 29,90*** sFr 24,90*/öS 299,-*

Bewegte Grafik per Farbdurchlauf – Diashow für Anspruchsvolle

Es ist vollbracht! Die besten Cycling-Grafiken können nun alle grafikbegeisterten Leser hautnah erleben. Die schönsten Bilder des **»Color-Cycle«-Wettbewerbs**, das im Amiga-Magazin (Ausgabe 3/88 Seite 142) gestartet wurde, präsentieren wir Ihnen auf zwei randvoll bespielten Disketten.
Die Bilder können entweder mit dem enthaltenen Diashow-Programm angesehen oder mit jedem gängigen IFF-Malprogramm (zum Beispiel Deluxe Paint II) geladen werden. Lassen Sie sich die faszinierenden Computer-Bilder nicht entgehen.
Zwei Disketten für Amiga.

Bestell-Nr. 49901 **DM 29,90*** sFr 24,90*/öS 299,-*

Die Wiederbelebung für die C64-Peripherie

Viele Amiga-Besitzer haben noch einen C64 mit Peripheriegeräten zu Hause stehen. Mit ein bißchen Hard- und Software können Sie diese zu neuem Leben erwecken und Ihre Daten so weiterbenutzen. Dabei ist die Bedienung wirklich einfach.
Der fertig aufgebaute IEC-Handler erlaubt es, alle C64-Geräte wie die Floppy 1541 oder 1571, Commodore-MPS-Drucker und natürlich auch den C64 (zur Datenübertragung) am Amiga zu betreiben.
Das Gesamtpaket besteht aus der fertig aufgebauten Platine mit Verbindungskabel, der Treibersoftware auf 3 1/2"-Diskette sowie einer entsprechenden Dokumentation.

Bestell-Nr. 39101 **DM 79,-*** sFr 71,-*/öS 790,-*

Bestellungen bitte nur gegen Vorauskasse bei:
Markt&Technik Verlag AG
– Buchverlag –
IEC-Handler
Hans-Pinsel-Straße 2, 8013 Haar bei München

Amiga 3/88 Bildschirmfüllende Boot-Bilder mit allen Extras

BootGirl: Fantastische Bilder sofort nach dem Reset. Bis zu 32 Farben mit Color-Cycling. Die Bilder können auch bildschirmfüllend ohne Rand sein. Ein absolutes Muß für jeden Amiga-Besitzer.
CassCover: Selbstgedruckte Kassettenhüllen geben Ihnen den richtigen Überblick. Einfache Bedienung macht das Eingeben und Ausdrucken zur wahren Freude.
Command: Das Programm ermöglicht die Steuerung des Aztec-C-Compilers mit der Maus. Keine langen Eingaben per Tastatur, sondern ein einziger Mausklick startet nun die Übersetzung.
VideoText: Ein unentbehrliches Werkzeug für alle Video-Fans, die ihren eigenen Vorspann mit dem Amiga generieren wollen. Laufbänder, verschiedene Schriften und IFF-Bilder sind nur einige Stichpunkte, die das Programm so interessant machen.
3 1/2"-Diskette für Amiga

Bestell-Nr. 48803 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 12/87 Super-Kopierprogramm mit viel Komfort

DCopy: Unser Programm des Monats, ein Kopierprogramm, das alles bietet, was man sich nur wünschen kann. Einige Fähigkeiten: Bis zu vier Laufwerke werden verwendet, Mehrfachkopien, abschaltbares Verify und vieles mehr.
SpeedHc: Eine sehr schnelle Hardcopyroutine für Schwarzweißausdrucke mit höchster Qualität. Leicht an andere Drucker anzupassen.
Sternenhimmel: Ein unentbehrliches Werkzeug für alle Himmelsbeobachter. Das Programm zeigt alle Sterne und Planeten von jedem beliebigen Punkt der nördlichen Hemisphäre.
Checkie42: Der Checksummer für alle Programmiersprachen von Assembler über Basic bis zu C. Ab dieser Ausgabe finden Sie bei jedem Listing die Prüfziffern.
Joy: Ein sehr kurzes und schnelles C-Programm zur Abfrage des Joysticks. Es ist leicht in eigene Programme einzubinden.
Amiga-Shell: Ein C-Programm, das Komfort ins CLI bringt. Editieren der Befehlszeile, Funktionstastenbelegung und Aliasnamen sind nur einige Fähigkeiten dieses fantastischen Programms.
3 1/2"-Diskette für Amiga

Bestell-Nr. 48705 **DM 29,90*** sFr 24,90*/öS 299,-*

* Unverbindliche Preisempfehlung

Übrigens: Mit den Gutscheinen aus dem »Super-Software-Scheckheft« für DM 149,– können Sie sechs Software-Disketten Ihrer Wahl aus dem Programm-Service-Angebot der Zeitschriften

PC Magazin	Happy-Computer-Sonderheft	Computer persönlich
PC Magazin Plus	Amiga-Magazin	64'er-Magazin
Happy-Computer	Amiga-Sonderheft	64'er-Sonderheft

bestellen – egal, ob diese DM 29,90 oder DM 34,90 kosten. Das Scheckheft können Sie per Verrechnungsscheck oder mit der eingehefteten Zahlkarte direkt beim Verlag bestellen.
Kennwort: Software-Scheckheft, Bestell-Nr. 39100.

Flags, die alle im Headerfile »graphics/view.h« (gleich nachsehen!) definiert sind, kann beispielsweise die Grafikauflösung festgelegt werden.

Die Flags, es existieren zur Zeit neun, können beliebig miteinander kombiniert werden. Man sollte jedoch auf die Funktionsfähigkeit der Kombination achten, denn es wird keine Plausibilitätskontrolle von Seiten des Betriebssystems durchgeführt. Hier die möglichen Flags:

»HIRES« ist die Abkürzung für »HIgh RESolution« und bedeutet hohe Auflösung. Dies bezieht sich jedoch nur auf die horizontale Bildschirmauflösung. Ist HIRES gesetzt, beträgt diese 640 Punkte, ansonsten die Hälfte, also 320 Punkte.

»LACE« verdoppelt die vertikale Auflösung von 256 auf 512 Punkte. Dies erreicht man durch eine besondere Technik, die auch beim Farbfernseher eingesetzt wird: dem Zeilensprungverfahren.

Das darzustellende Bild zerteilt sich in zwei Halbbilder, »LongFrame« und »ShortFrame«. LongFrame wird auf allen ungeraden Zeilen (also Zeile 1,3,5 etc.), ShortFrame auf allen geraden Zeilen (2,4,6 etc.) dargestellt. Die eigentliche Bildschirmfrequenz vom 50 Hz verringert sich dadurch auf 25 Hz und kommt somit der Trägheitsgrenze des menschlichen Auges sehr nahe, was sich mit einem ziemlich starken Flimmern bemerkbar macht.

»SPRITES« müssen Sie setzen, wenn Sie auf Ihrem Screen Hardware-Sprites darstellen möchten.

»DUALPF« Zwei übergroße Grafiken können »übereinander« dargestellt werden. Diese Bilder können gegeneinander verschoben werden und an transparenten Stellen der oberen Grafik wird die untere sichtbar. Jedes Playfield kann aus maximal drei Bit-Ebenen bestehen, so daß sich für jedes acht Farben ergeben. Die Farbregister 0 bis 7 sind für das erste, die Register 8 bis 15 für das zweite Playfield zuständig, wobei die Farben 0 und 8, also die jeweils ersten eines Playfields, transparent sind. Wer sich für diesen interessanten Darstellungsmodus interessiert, sollte sich unser Demoprogramm auf Seite 145 (»Doppeltes Spiel«) anschauen.

»EXTRA_HALFBRITE« bringt zum ersten Mal die sechste Bit-Ebene ins Spiel. Obwohl uns im »LoRes«-Modus nur 32 Farbregister zur Verfügung stehen, können wir im EXTRA_HALFBRITE-Modus 64 Farben ansprechen und darstellen. Beachten Sie hierzu auch Listing 7 (EHB.c). Addieren Sie zu einer Farbregisternummer den Wert 32, wird deren nach rechts verschobener Inhalt als Farbe dargestellt. Die Farben 32 bis 63 entsprechen also den Farben 0 bis 31, sind jedoch nur halb so hell. Vermutlich leitet sich so der etwas merkwürdig klingende Name dieses Flags ab.

```
/*
\ \  EHB.c - Extra-Halfbright Demo
*/
#include "TOOL.h"
#define VP &(Screen->ViewPort)
#define RP &(Screen->RastPort)
struct Screen *Screen;
VOID error(meldung)
STRPTR meldung;
{
if(meldung) printf("ERROR: %s",meldung); /* Fehlermeldung */
if(Screen) CloseScreen(Screen); /* Screen schließen */
CloseLib(); /* Libraries schließen */
exit(0);
}
VOID main()
{
REGISTER COUNT i,j,ypos=256;
if(OpenLib()){  /* Libraries öffnen */
error("Kann Libraries nicht öffnen!\ n"); /* Fehlermeldung */
}
if(!(Screen=GetScreen(ypos,6,EXTRA_HALFBRITE,"EHB-SCREEN Demo"))){
error("Kann Screen nicht öffnen !\ n");
}
for(i=0,j=0;i<16;i++,j+=2){  /* 32 Farben definieren */
SetRGB4(VP, i,j,j+1,i);
SetRGB4(VP,16+i,i,i/2,j);
}
for(i=0;i<32;i++){
SetAPen(RP,i); /* Zeichenfarben 0 bis 31 festlegen */
RectFill(RP,10*i, 12,10*i+9,255); /* Rechteck zeichnen */
SetAPen(RP,i+32); /* Zeichenfarben 32 bis 63 (!) festlegen */
RectFill(RP,10*i,100,10*i+9,155);
}
while(ypos-->=0) MoveScreen(Screen,0,-1); /* Screen verschieben */
Delay(250); /* etwas warten */
error(0); /* Programmende ohne Fehlermeldung */
}
```

Listing 7. EHB.c demonstriert den Extra-Halfbrite Modus

Um die Farben der Farbregister festzulegen, existiert in der Grafik-Bibliothek eine Routine namens »SetRGB4()«. Hier der Aufruf:

```
SetRGB4(VP,FR,rot,grün,
blau);
```

»VP« ist ein Zeiger auf die ViewPort-Struktur (siehe unten). »FR« ist die Nummer des Farbregisters, dessen Inhalt sich verändern soll. Die Parameter »rot«, »grün« und »blau« repräsentieren die Farbanteile, aus denen sich die Farbe zusammensetzt.

Der Farbtopf

Erlaubt sind Werte von 0 bis 15 für jeden Farbanteil. Es handelt sich hier, wie beim Sonnenlicht, um eine additive Farbmischung, das heißt die Maximalwerte der drei Grundfarben ergeben weiß.

In der Hauptfunktion unseres EHB-Demos definiert SetRG B4() 32 Farben innerhalb einer Schleife. Wir lernen später noch eine andere Funktion kennen, mit deren Hilfe man die komplette Palette eines Screens festlegen kann.

Die Wirkung der 5. und 6. Bitebene bei HAM

Ebene 5	Ebene 6	modifizierter Anteil	Bitnummer im Farbregister
0	0	—	—
0	1	blau	0 bis 3
1	0	rot	8 bis 11
1	1	grün	4 bis 7

Tabelle 5. Im HAM-Modus entscheiden die Bit-Ebenen 5 und 6, welcher Farbanteil modifiziert werden soll

Wenn wir schon dabei sind, die Anzahl der zur Verfügung stehenden Farbenpracht auf die Spitze zu treiben, wie wäre es mit 4096 Farben?

Dies ist tatsächlich realisierbar, allerdings – wie der »EXTRA_HALFBRITE«-Modus – nur in der niedrigen Auflösung und mit Hilfe der sechsten Bitplane. Eine weitere Einschränkung besteht darin, daß die Farbe eines gesetzen Punktes, zu einem Drittel von der Farbe des linken Nachbarpunktes abhängig ist. HAM bedeutet nichts anderes als »Hold And Modify« (halte und verändere). Im Klartext: zwei Anteile einer Farbe bleiben bestehen und der dritte wird verändert. Welchen der drei Anteile einer Farbe, rot, grün oder blau, Sie verändern möchten, teilen Sie mit den Bit-Ebenen 5 und 6 mit. Tabelle 5 zeigt die möglichen Kombinationen. Sind beide Bits auf 0 gesetzt, wird die Nachbarfarbe unverändert übernommen.

»PFBA« Wie im Dual-Playfield-Modus wird auch hier mit zwei hintereinanderliegenden Screens gearbeitet. Allerdings müssen diese im PFBA-Modus gleich groß sein. Während die Grafik der vorderen Bitmap sichtbar ist, kann beispielsweise eine andere unsichtbar im Hintergrund aufgebaut werden. So entstehen zum Beispiel bei Flugsimulatoren, trotz der relativ komplexen Grafik, fließende Bewegungen. Man nennt diese Technik »Double Buffering«.

»GENLOCK_VIDEO« ersetzt die Hintergrundfarbe durch das Signal des Genlock-Video-Interface. Endlich brauchen Sie nicht mehr auf »Wetten daß« zu verzichten, während Sie den neuen C-Kurs durcharbeiten!

»VP-HIDE« In diesem letzten, etwas kuriosen Modus, wird der Screen nicht dargestellt.

Type ist die nächste Variable der NewScreen-Struktur. Wenn Sie Ihren ganz persönlichen Screen öffnen möchten, setzen Sie hier das Flag »CUSTOMSCREEN«. Mit »WBENCHSCREEN« erhalten Sie eine Kopie des Workbenchscreen. Das hat den Vorteil, daß darauf die Systemmeldungen, die Ihnen der Amiga mittels Requester überbringen möchte, sichtbar sind. Bei der Verwendung eines Customscreens schaltet sich, wenn der Bediener Ihres Programms zum Beispiel die nicht vorhandene Diskette in Laufwerk DF0: ansprechen möchte, der im Hintergrund liegende

Workbenchscreen mit dem entsprechenden Requester nach vorne. Wenn Sie dem Anwender Ihres Programms nicht zumuten wollen, daß er den Workbenchscreen selbst wieder nach hinten klickt, muß Ihr Customscreen vom Programm aus wieder nach vorne gebracht werden.

Es besteht auch die Möglichkeit, sich ein komplett eigenes Display zu erstellen. Sie müssen dann eine eigene BitMap Struktur definieren (siehe Listing 15) und hier, unter Type, das Flag »CUSTOMBITMAP« setzen (Beachten Sie aber, daß diese Bitmap auch mit der Screen-Struktur verknüpft werden muß).

Fonts Diese Variable erwartet einen Zeiger auf eine Text Attr-Struktur. Sie können so einen beliebigen Zeichensatz verwenden. Ist man jedoch mit dem eingestellten Zeichensatz zufrieden, so setzt man Fonts gleich NULL.

Die TextAttr-Struktur ist sehr einfach aufgebaut, wir werden uns später intensiv damit befassen. Ganz besonders Neugierige verweisen wir auf das Headerfile »graphics/text.h«.

Title Hier können Sie den Text, der später in der Titelleiste des Screens erscheinen soll, eintragen. Wünschen Sie keinen Screen-Titel, so geben Sie einfach NULL ein.

Gadgets Diese Variable ist zur Zeit noch ungenutzt, weshalb sie immer auf NULL gesetzt sein sollte.

CustomBitMap, die letzte Variable der NewScreen-Struktur, erwartet einen Zeiger auf eine selbstdefinierte BitMap-Struktur. Eine BitMap ist eine Datenstruktur, die die Speicherbereiche der Bit-Ebenen enthält. Wenn Sie mit einer selbstdefinierten BitMap-Struktur arbeiten möchten, muß das Flag »CUSTOMBITMAP« der Variablen Type (siehe oben) gesetzt sein.

Definieren Sie CustomBitMap als NULL, wenn Sie lieber keine eigene BitMap-Struktur definieren wollen. Achten Sie darauf, daß in diesem Fall die Variable Type der NewScreen-Struktur nicht auf CUSTOMBITMAP gesetzt ist.

Wenn Ihnen die Datentypen der Variablen innerhalb der NewScreen-Struktur etwas merkwürdig erscheinen, sollten Sie sich einmal das Headerfile »exec/types.h« anschauen. Dort sind alle speziellen Amiga-Datentypen definiert. Es handelt sich jedoch nicht um gänzlich neue Typen, sondern den Standard-Datentypen wurden

```
struct Screen
{
struct Screen *NextScreen;
struct Window *FirstWindow;
SHORT LeftEdge, TopEdge;
SHORT Width, Height;
SHORT MouseY, MouseX;
USHORT Flags;
UBYTE *Title;
UBYTE *DefaultTitle;
BYTE BarHeight, BarVBorder, BarHBorder;
BYTE MenuVBorder, MenuHBorder;
BYTE WBorTop, WBorLeft, WBorRight, WBorBottom;
struct TextAttr *Font;
struct ViewPort ViewPort;
struct RastPort RastPort;
struct BitMap BitMap;
struct Layer_Info LayerInfo;
struct Gadget *FirstGadget;
UBYTE DetailPen, BlockPen;
USHORT SaveColor0;
struct Layer *BarLayer;
UBYTE *ExtData;
UBYTE *UserData;
} ;
```

Listing 5. Die Screen-Struktur enthält wichtige Informationen über einen bereits geöffneten Screen

neue Namen verliehen. Es steht nichts im Wege, Ihr geliebtes »short« anstelle des neuen »COUNT« weiterzuverwenden.

Der erste eigene Screen

Ob Sie die einzelnen Komponenten der NewScreen-Struktur durch eine statische Initialisierung oder innerhalb einer Funktion per Auswahloperator mit Werten bestücken, bleibt Ihnen überlassen. In unserem Headerfile TOOL.h geben wir der letzteren Methode den Vorzug.

Es hält uns aber in jedem Falle nichts mehr auf, unseren ersten eigenen Screen zu öffnen. Dabei hilft uns die Intuition-Funktion OpenScreen().

```
OpenScreen(&NewScreen);
```

Als Parameter dient ein Zeiger auf die NewScreen-Struktur, die das Aussehen des gewünschten Screens definiert. Als Returnwert liefert diese Funktion einen Zeiger auf eine weitere Struktur – die Screen Struktur (Listing 5).

Wie das Öffnen einer Library, erfolgt das Öffnen eines Screens immer nach dem gleichen Schema, eine Automatisierung mit Hilfe einer geeigneten Routine bietet sich an. Die erste Intuition-Funktion in Tool.h zeigt die fertige Lösung. GetScreen() übernimmt für uns die Initialisierung der NewScreen-Struktur und das Öffnen des neuen Screens.

Als Parameter dienen TopEdge, Depth, ViewModes und DefaultTitle. Die anderen Variablen der Struktur erhalten gängige oder, wie Width und Height, errechnete Werte. Somit gestaltet sich der Funktionsaufruf denkbar einfach:

```
GetScreen(t,d,m,ti);
```

Die Funktion liefert die Adresse einer Screen-Struktur. Mit dieser Struktur befassen wir uns noch im Anschluß ausführlich. Zuvor jedoch hat unsere Funktion noch in Listing 6 den ersten Test zu bestehen. Wir fordern gleich drei Bildschirme an und noch dazu mit unterschiedlichen Auflösungen. Jeder der drei GetScreen()-Aufrufe liefert die Adresse auf die Struktur des gerade zu öffnenden Screens. Wir weisen diese Werte den globalen Zeigern Screen1, Screen2 und Screen3 zu. Nach dem Öffnen der benötigten Libraries werden nacheinander die drei Bildschirme geöffnet.

Sollten die Libraries oder einer der Screens nicht zu öffnen sein, so erfolgt in der Funktion »error()« eine Fehlermeldung. Alle Screens und Libraries werden dort in umgekehrter Reihenfolge geschlossen.

Eine unbekannte Funktion schließt sich an. Delay(n) verzögert den Programmablauf um n/50 Sekunden. Lassen Sie das Bild dieser drei Screens ein wenig auf sich wirken. Ein Bildschirmblitz holt Sie in die Wirklichkeit zurück. Haben Sie bemerkt, daß nur der mittlere Screen geblinkt hat? Als Parameter von DisplayBeep() diente der Zeiger auf die Screen-Struktur des mittleren Bildschirms Screen2. Jetzt wird auch die Bedeutung dieses Zeigers klar: mit seiner Hilfe kann ein Screen wie mit einem Namen angesprochen werden. Dies ist auch bei der nächsten Funktion der Fall: CloseScreen(Screen1) schließt einzig und allein den Screen mit der zugehörigen Struktur mit der Adresse »Screen1«.

Nacheinander, mit einer kurzen Verzögerung, werden zuerst alle Screens und dann die Libraries geschlossen.

Vielleicht sollten wir, bevor wir die Screen-Struktur besprechen, einige Screen-Funktionen testen? Kein Problem. Die Variable »s« sei als Zeiger auf die Screen-Struktur definiert (struct Screen *s;). Die Funktionen OpenScreen() und CloseScreen() sind bereits besprochen, in diesem Zusammenhang interessieren zwei Workbench-Funktionen:

```
CloseWorkbench();
OpenWorkbench();
```

Der Workbenchscreen wird geschlossen beziehungsweise geöffnet. Beide Funktionen liefern den Wert 1 (True) wenn alles geklappt hat und 0 im anderen Fall. Parameter benötigen beide nicht.

```
MakeScreen(s);
```

Ein bereits geschlossener Screen, dessen Zeiger noch besteht, kann mit Hilfe dieser Funktion gerettet werden. Die Eingliederung des Screens in die Copperliste erfolgt im Anschluß durch den Aufruf der Funktion RethinkDisplay().

RemakeDisplay() frischt mit Hilfe dieser beiden Funktionen das gesamte Intuition-Display neu auf. Zuerst erfolgt mit jedem Screen der Aufruf von MakeScreen(). Im Anschluß daran aktualisiert RethinkDisplay() die Copperliste.

```
fehler=GetScreenData
(puffer,anzahl,type,
screen);
```

Diese Funktion kopiert eine Anzahl (»anzahl«) von Daten in einen im RAM gelegenen Puffer (»puffer«). Der Typ (»type«) kann »CUSTOMSCREEN« oder »WBENCHSCREEN« sein.

Geben Sie WBENCHSCREEN als Type an, wird das vierte Argument ignoriert. Im anderen Fall steht dort der Zeiger auf eine Screen-Struktur. Die Funktion kehrt im Fehlerfalle mit 0 zurück.

```
MoveScreen(s, px, py);
```

Der Screen »s« wird »py« Punkte verschoben. Durch die Angabe von positiven Werten

verschiebt sich der Bildschirm mit Inhalt nach unten, eine Bewegung nach oben erreichen Sie durch die Angabe von negativen Werten. Da ein Screen nicht in X-Richtung verschoben werden kann, hat »px« zur Zeit noch keine Bedeutung.

`ScreenToBack(s);`

Diese Funktion schaltet den Screen s hinter alle anderen, zu diesem Zeitpunkt geöffneten Screens. Das Gegenstück dazu ist ScreenToFront(s).

`ScreenToFront(s);`

Screen »s« wird als »vorderster« Screen dargestellt. Auch den Workbenchscreen können Sie nach vorne oder nach hinten schalten. Hierzu dienen die Funktionen WBenchToBack() und WBenchToFront(). Beide benötigen keine Parameter und liefern den Wert 1 (TRUE) wenn alles geklappt hat oder 0 wenn nicht.

`ShowTitle(s,modus);`

Entscheiden Sie mit 1 oder 0 als Modus, ob der Screen Titel auf einem Backdrop-Window (Kapitel Windows) erscheinen soll (1) oder nicht (0).

Zum Schluß soll noch etwas Farbe ins Spiel. SetRGB4() haben Sie bereits kennengelernt. Die Funktion

`LoadRGB4(&VP,&Palette, n);`

erlaubt die Definition mehrerer Farben auf einmal. VP ist dabei die Adresse des ViewPorts und n die Anzahl der Farbregister, die Sie verändern möchten. Palette ist die Adresse eines WORD-Arrays, welches die Farbwerte, »n« Stück an der Zahl, enthält. Das Gegenstück zu SetRGB4() heißt GetRGB4 (&CM,nr). Die Funktion liest den Wert nr aus der ColorMap CM und liefert den Wert des Farbregisters. Die ColorMap finden Sie in der ViewPort-Struktur: »ViewPort.ColorMap«. Die einzelnen Farben in R, G und B muß sich der Anwender selbst errechnen – oder einfach Tool.h verwenden, denn da ist eine Funktion definiert, die das übernimmt.

`nr=GetCol(&VP,nr, &r, &g,&b);`

Die Funktion liefert, ebenso wie GetRGB4(), den Inhalt des Farbregisters »nr«. Darüber hinaus werden aber noch die Variablen r, g und b, deren Adressen Sie übergeben, mit den Rot/Grün/Blau-Werten belegt. Der Aufruf ist dem von Set RGB4() angepaßt. Daher ist VP die Adresse des ViewPorts und nr, wie erwähnt, die Nummer des Farbregisters.

Testen Sie ruhig einmal alle Funktionen, bevor es jetzt mit der Screen-Struktur weitergeht.

Immer aktuell – die Screen-Struktur

Nachdem Sie den »Antrag« (NewScreen-Struktur) für einen neuen Bildschirm abgegeben haben, erhalten Sie ein ausführliches »Informationsblatt« zurück, die Screen-Struktur (Listing 5). Die einzelnen Variableninhalte der Struktur können Sie wie gewohnt mit dem Inhaltsoperator lesen:

```
struct Screen *S;
SHORT xpos;
...
xpos=S->*MouseX;
```

In diesem Fragment übertragen wir die aktuelle x-Position des Mauszeigers in die Variable xpos. Betrachten wir uns die Screendaten etwas genauer:

NextScreen ist ein Zeiger auf den nächsten Screen. So sind alle Screens verkettet, wenn nur ein einziger bekannt ist, kann man per »Durchhangeln« mit dem Inhaltsoperator alle anderen erreichen. Diese Art der Verkettung finden Sie bei jeder wichtigen Amiga-Datenstruktur.

Existiert kein Screen mehr, enthält NextScreen den Wert 0.

FirstWindow Hier finden Sie einen Zeiger auf das erste Fenster auf dem Screen. Da auch alle Windows miteinander verkettet sind, können Sie mit dieser Information auch auf alle anderen Fenster dieses Screens zurückgreifen.

LeftEdge, TopEdge, Width und **Height** sind die Screenpositionen und Screendimensionen aus der NewScreen-Struktur.

MouseX und **MouseY** halten ständig die Position des Mauszeigers parat. Dadurch kann diese jederzeit abgefragt werden. Bezugspunkt ist die linke obere Ecke des Screens (x=0 und y=0).

Flags neben den Flags, die Sie selbst in der NewScreen-Struktur gesetzt haben, finden Sie hier noch »SHOWTITLE« und »BEEPING«. SHOWTITLE verändert sich mit einem Aufruf der Funktion »ShowTitleScreen()«. BEEPING ist gesetzt, wenn der Bildschirm blinkt.

Title ist ein Zeiger auf den Titeltext des Screens.

DefaultTitle ist ein Zeiger auf einen Titel für den Fall, daß ein Window ohne Screentitel aktiv ist.

Die nächsten Variablen geben die Abmessungen der Titelleiste für den Screen und die darauf befindlichen Windows an.

Font enthält einen Zeiger auf die in der NewScreen-Struktur gewählte TextAttr-Struktur.

ViewPort. Hier finden Sie eine weitere kleine Struktur (ausnahmsweise nicht den Zeiger darauf) – die ViewPort-Struktur. Sie enthält unter anderem einen Zeiger auf den nächsten ViewPort, einen Zeiger auf die ColorMap, die Dimensionen des zu verwaltenden Displayausschnittes, Informationen über die Bitplanes, die in der NewScreen-Struktur festgelegten ViewModi und verschiedene Listen für den Copper.

Sie sehen, der ViewPort ist maßgeblich an der Gestaltung des Displays beteiligt. Ein Display kann aus einem oder mehreren ViewPorts bestehen.

Dabei ist zu beachten, daß diese nicht nebeneinander dargestellt werden können und daß zwischen den einzelnen ViewPorts mindestens eine (Pixel-) Zeile frei bleiben muß. All dies erinnert an den Screen und tatsächlich ist es so, daß man den Screen einfach nur als den sichtbaren Teil des ViewPorts bezeichnen könnte.

RastPort Auch die RastPort-Struktur ist in der Screen-Struktur nicht als Zeiger, sondern höchstpersönlich vertreten. Er enthält, als Verwalter einer Zeichenebene, verschiedene Zeiger auf Grafikgrundelemente, wie beispielsweise die Layerstrukturen oder die Bitmapstrukturen. Weiter findet man dort die Vorder- und Hintergrundfarbe, die Zeichen- und Zeichensatzmodi, die Position und Maße des Grafikcursors und vieles mehr. Wir kommen im Zusammenhang mit den Windows noch einmal auf den RastPort zu sprechen.

BitMap An dieser Stelle ist die Datenstruktur abgelegt, die die Adressen der Speicherbereiche des Screens, der Bit-Ebenen, beinhaltet.

LayerInfo ist eine weitere Datenstruktur der Screen-Struktur. Sie informiert über die sogenannten Layers, die Grundelemente der Amiga-Windows.

FirstGadget Für den Screen existieren bisher nur die beiden Systemgadgets. Mit dem ersten kann man ihn nach vorne oder in den Hintergrund klicken, mit

```
/*
\ \ 3SDemo.c - Demo mit drei Screens
*/
#include "TOOL.h"
struct Screen *Screen1, *Screen2, *Screen3;
VOID error(meldung)
STRPTR meldung;
{ /* Fehlermeldung ausgeben und Programm beenden */
printf("\ nKann %s nicht öffnen!\ n",meldung);
if(Screen3) CloseScreen(Screen3);
if(Screen2) CloseScreen(Screen2);
if(Screen1) CloseScreen(Screen1);
CloseLib();
exit(0);
}
VOID main()
{
if(OpenLib()){ /* Libraries öffnen */
error("Libraries");
}
if(!(Screen1=GetScreen(50,2,LACE,"LACE"))){ /* Screen öffnen */
error("LACE-Screen");
}
if(!(Screen2=GetScreen(100,2,HIRES,"HIRES"))){
error("HIRES-Screen");
}
if(!(Screen3=GetScreen(400,2,HIRES|LACE,"HIRES|LACE"))){
error("HIRES|LACE-Screen");
}
Delay(200); /* ca. 4 Sekunden warten ... */
DisplayBeep(Screen2); /* Bildschirmblitz */
Delay(50);
CloseScreen(Screen1); /* Screen schließen */
Delay(50);
CloseScreen(Screen2);
Delay(50);
CloseScreen(Screen3);
CloseLib(); /* Libraries schließen */
}
```

Listing 6. Das »Screendemo« öffnet drei verschiedene Screens

dem anderen in der Y-Richtung frei verschieben. In einer späteren Version des Betriebssystems wird hier der in der NewScreen-Struktur vereinbarte Zeiger auf das erste Benutzergadget zu finden sein.

Detail- und **BlockPen** sind ebenfalls in der NewScreen-Struktur definiert.

SaveColor0 Hier sichert Intuition die Hintergrundfarbe (Farbregister 0) während eines Bildschirmblitzes.

BarLayer ist ein Zeiger auf die Layer-Struktur der Titelleiste.

Ext- und **UserData** enthalten Zeiger auf eigene Daten, die den Screen betreffen.

Doch vorerst genug der grauen Theorie. Bevor es mit den Windows weitergeht, wollen wir unser Wissen über Screens noch mit einigen Übungen vertiefen.

Auch auf den folgenden Seiten werden Sie nach jedem größeren Kapitel auf solche Übungen treffen. So können Sie selbst Ihren Wissensstand kontrollieren und wenn nötig noch vertiefen. Zuerst untersuchen wir gemeinsam den Extra-Halfbrighte-Screenmodus. In diesem Modus stehen 64 Farben zur Verfügung. Um diese auch auf dem Bildschirm sichtbar zu machen, müssen wir etwas vorgreifen und zwei noch nicht besprochene Grafikroutinen benutzen:

```
SetAPen(RP,col);
RectFill(RP,x1,y1,
x2,y2);
```

SetAPen() legt die Zeilenfarbe (col) für den angegebenen RastPort (RP) fest. Die andere Funktion, RectFill(), zeichnet im RastPort RP ein mit der eben festgelegten Farbe gefülltes Rechteck mit den angegebenen Koordinaten. Mehr über diese Funktionen erfahren Sie weiter unten im Grafikteil.

Beleuchten wir nun unser »EHB-Demo« (Listing 7) etwas genauer: Nachdem Tool.h eingebunden ist, erfolgt die Vereinbarung zweier Symbole. VP ist die Adresse der ViewPort Struktur, RP die der RastPort-Struktur des Screens. Die Klammern sind in beiden Fällen nicht unbedingt notwendig.

Der Zuweisungsoperator (->) hat ohnehin die höhere Priorität gegenüber dem Adreßoperator (&). Sie verdeutlichen dem Leser Ihres Programms, daß hier ausdrücklich die Adresse der jeweiligen Strukturen verlangt wird.

In der Hauptfunktion main() öffnen wir die Libraries und den Screen. Falls hierbei ein Fehler auftreten sollte, gibt »error()« eine Fehlermeldung aus und bricht das Programm ab.

Es folgen in main() zwei »for«-Schleifen. In der ersten belegen wir die 32 Farbregister des Screens mit Werten. Im Rumpf der nächsten Schleife zeichnen wir 32 Rechtecke mit Screenhöhe, in den Farben 0 bis 31 und weitere 32 Rechtecke, in die Screenmitte, in den Farben 32 bis 63. Der Screen ist jedoch noch nicht sichtbar. Denn beim Öffnen legte die Variable ypos dessen oberes Ende auf die Position 256. Erst jetzt sorgt »MoveScreen()« in einer »while«-Schleife für eine kontinuierliche Bewegung des Screens von unten nach oben. Zum Abschluß werden, nach einer kleinen Pause, Screen und Libraries geschlossen. Versuchen Sie doch jetzt einmal, einen eigenen Screen zu öffnen, oder sogar zehn Stück, jeder mit einer anderen Farbenpalette. In der Titelleiste könnten die Screens durchnumeriert sein.

Streichen Sie die Delay()-Funktion in EHB.c und ersetzen Sie sie durch eine Routine, die wartet, bis der Anwender den Mauszeiger in die linke obere Ecke bewegt. Testen Sie die View-Modi. Können Sie mehr als 64 Farben auf einen Screen bringen? Verändern Sie die besprochenen Programme nach eigenen Wünschen, experimentieren Sie damit. Sie werden über die Ergebnisse überrascht sein.



Neben dem Screen stellt uns Intuition noch einen weiteren elementaren und zugleich flexiblen Baustein zur Verfügung: gemeint ist das Window. Windows auf einem Screen lassen sich in gewisser Weise mit losen Arbeitsblättern auf einem Schreibtisch vergleichen. So kann man sie beliebig positionieren oder auch in ihrer Reihenfolge verändern.

Allerdings ist die Größe eines Windows, anders als die eines Papierbogens, nicht genormt. Im Gegenteil, oft kann der Anwender sein Arbeitsfeld sogar nach Bedarf vergrößern oder verkleinern. Nehmen wir als Beispiel das CLI-Window. Dieses ist mit vier sogenannten Systemgadgets ausgerüstet (siehe Bild 2). Mit Hilfe des ersten Gadgets, der Titelleiste, auch »Dragbar« genannt, läßt sich das CLI-Fenster nach Wunsch verschieben.

Das rechte Ende der Titelleiste bilden die beiden Tiefengadgets. Per Mausklick verändert man die Reihenfolge der Windows. In der rechten unteren Ecke schließlich, ist das »WINDOWSIZE«-Gadget montiert, von wo aus Sie Ihr CLI-Fenster auf die gewünschte Größe »ziehen« können.

Im CLI geben Sie Befehle über die Tastatur ein. AmigaDOS führt diesen Befehl aus oder quittiert eine falsche Eingabe mit einer entsprechenden Fehlermeldung. Sie sehen daran, daß auch die Ein- und Ausgaberoutinen über die Windows laufen. Man bezeichnet sie daher gelegentlich als die Schnittstelle zum Anwender.

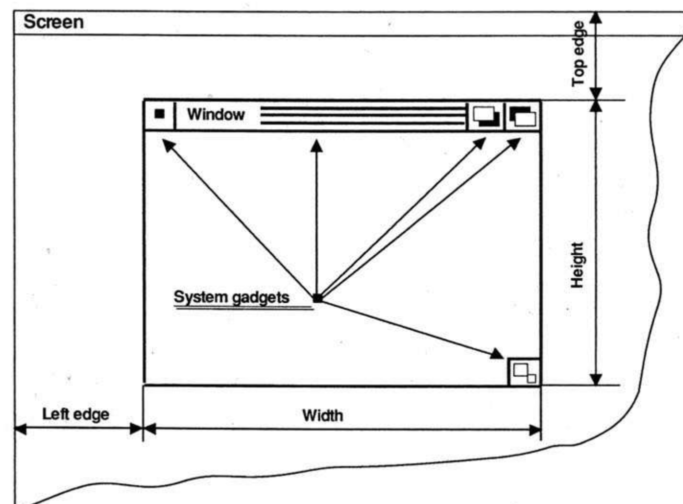


Bild 2. Die Lage eines Windows mit den fünf Systemgadgets auf einem Screen

Mit dem CLI-Kommando »NewCLI« kann man vom CLI aus weitere Fenster öffnen. Die Anzahl ist nur vom zur Verfügung stehenden Speicherplatz abhängig.

Dem Programmierer stellt das Betriebssystem grundsätzlich zwei Möglichkeiten zur Auswahl, um ein eigenes Fenster vom Programm aus zu öffnen. Die eine Möglichkeit führt über Intuition, mit der Funktion OpenWindow(). Die zweite Möglichkeit nutzt eine Besonderheit des AmigaDOS, die Verwaltung sogenannter Fenster-Devices über das Device »CON:«. Betrachten wir zunächst einmal Möglichkeit Nummer 2.

Listing 8 zeigt, wie einfach vom Programm aus ein neues CLI-Fenster geöffnet wird, wir brauchen noch nicht einmal unser Tool. Das fertig compilierte Programm öffnet ein neues CLI-Fenster und führt darin den eventuell mitgegebenen Kommandoparameter als CLI-Kommando aus; Beispiel:

```
CLI dir
```

Die Funktion »Open()«, mit den beiden Parametern »Filename« und »Modus«, öffnet uns dieses Fenster automatisch, wenn wir es mit dem Filenamen »CON:« ansprechen. CON: ist zuständig für Bildschirm und Tastatur und somit für uns genau das Richtige:

```
FH=Open("CON:X/Y/B/H/T",
Modus);
```

Hinter der Angabe »CON:« sind die gewünschte Fensterposition (X und Y), dessen Dimensionen (B und H für Breite und Höhe) und der Text für die Titelleiste aufgeführt.

Die möglichen Modi, die als zweiter Parameter folgen, sind im Headerfile »libraries/dos.h« festgelegt. Wir verwenden hier »MODE_NEWFILE«, da wir mit »Open()« ein neues »Bildschirm-File« sprich AmigaDOS-Window öffnen möchten.

»Open()« kehrt, wenn alles klappt, mit einem Zeiger auf eine FileHandle-Struktur zurück. Diese Struktur repräsentiert die gerade geöffnete Datei, in unserem Fall also das neu geöffnete Fenster. Im Fehlerfalle liefert die Funktion den Wert 0. Der nun folgene Start von »Exe-

```
/*
\\  CLI-Window öffnen
*/
#include <libraries/dos.h>

VOID main(argc,argv)
   int argc;
   char *argv[];
{
   struct FileHandle *FH;

   FH=Open("CON:0/0/639/100/CLI-Window",MODE_NEWFILE);
   Execute((argc>1?argv[1]:""),FH,0);
   Close(FH);
}
```

Listing 8. Die Funktion Open() öffnet ein einfaches CLI-Window

cute()« führt das beim Aufruf mitgegebene CLI-Kommando aus. Voraussetzung ist allerdings, daß sich im Verzeichnis C Ihrer Systemdiskette das File »RUN« befindet. Execute verlangt drei Parameter:

`Execute(&K,&FH_in, &FH_out);`

»K« steht für den Zeiger auf den Namen des CLI-Kommandos, welches »Execute()« starten soll. Natürlich könnte an dieser Stelle auch ein gewöhnlicher Filename stehen. »FH__in« und »FH__out« sind Zeiger auf die FileHandle-Struktur des Ein- beziehungsweise Ausgabefiles (oder -Einheit). Wir kommen zu einem späteren Zeitpunkt noch einmal auf Execute() zurück.

Die Funktion »Close()« schließt das Fenster erst, wenn man über die Tastatur das Kommando »EndCLI« eingibt. Auch dieses, auf einfachste Art erzeugte Window, kann vom Bediener verschoben, verkleinert und vergrößert werden, ohne daß wir dies im Programm berücksichtigen mußten. Die Verwaltung der vier besprochenen Gadgets übernimmt Intuition selbst.

Es existiert jedoch noch ein fünftes Systemgadget, das CloseGadget. Es befindet sich, wenn aktiviert, in der linken oberen Ecke des Windows. Intuition schließt jedoch ein Fenster mit CloseGadget nicht selbständig, sondern teilt lediglich dem Programm mit, wenn der Anwender das Gadget anwählt. Diese Benachrichtigung geschieht über den sogenannten »MessagePort«, abgekürzt IDCMP. Doch hierzu lesen Sie weiter unten mehr.

Intuition benötigt, wie beim Screen auch, eine besondere Struktur, um ein Window korrekt verwalten zu können. Diese Window-Struktur enthält alle wichtigen Daten eines Windows und wird ständig aktualisiert. Wie wir gleich noch sehen werden, liefert der Aufruf der Intuition-Funktion »OpenWindow()« einen Zeiger auf diese Struktur. Die Window-Struktur eines CON:-Fensters müssen wir uns jedoch selbst beschaffen. Das Studium der IntuitionBase-Struktur zeigt uns eine einfache Lösung dieses Problems für gerade aktive Windows: IntuitionBase enthält nämlich unter anderem einen Zeiger auf das aktive Window. Da unser DOS-Window nach dem Öffnen aktiv ist, können wir diese Information verwerten:

```
struct Window *win;
win=IntuitionBase->
ActiveWindow;
```

Die Zugriffsmöglichkeit auf ein gerade aktives Fenster ist so wichtig, daß wir unser Headerfile Tool.h mit der obigen Lösung ergänzt haben. Vor deren Verwendung muß natürlich die Adresse der IntuitionBase-Struktur bekannt sein. Wir prüfen das zur Sicherheit in einem bedingten Ausdruck.

Das fertige Makro heißt »ACTIVE__WINDOW« (siehe Listing 1). Da auch der aktive Screen in IntuitionBase geführt wird, ist es sinnvoll ein gleichartiges Makro auch für Screens zu definieren: »ACTIVE__SCREEN«.

Listing 9 demonstriert die Anwendung der Makros. Achten Sie darauf, daß Sie **vor** deren Verwendung die Funktion OpenLib() aufrufen, so daß IntuitionBase auch tatsächlich bekannt ist. Ansonsten liefern ACTIVE__WINDOW und ACTIVE__SCREEN einen Nullpointer.

Zur Funktion des Programms: »Max.c« setzt das aktive Window in die linke obere Bildschirmecke und bringt es auf die durch den Screen begrenzte maximale Größe. MoveWindow() und SizeWindow() ersetzen die Funktionen der Systemgadgets WINDOWDRAG und WINDOWSIZE. Als Parameter dienen Werte der Screen-Struktur und der Window-Struktur (siehe unten).

Das Arbeiten mit AmigaDOS-Windows mag für einfache Anwendungen ausreichen. Ein Nachteil ist, daß die Windows immer vorgefertigt sind. Nach dem Öffnen hat das Fenster verschiedene Eigenschaften, die vielleicht nicht immer erwünscht sind. Andere, wie zum Beispiel die Möglichkeit zum Arbeiten mit Menüs, fehlen und müssen im nachhinein umständlich eingefügt werden.

Wie weiter oben bereits angedeutet, bietet uns aber das Betriebssystem einen Weg, ein eigenes, individuelles Fenster zu definieren.

Intuition verlangt hierzu von uns lediglich eine Art »Wunschzettel« in Form einer Struktur, in der wir das neue Window genau beschreiben. Sie sind durch die Arbeit mit Screens bereits mit dieser Arbeitsweise vertraut; das Öffnen eines Windows geschieht analog zum Öffnen eines Screens.

Bei einem Window übergeben wir der »OpenWindow()«-Funktion einen Zeiger auf die NewWindow-Struktur. Zurückgeliefert bekommen wir einen Zeiger auf die Window-Struktur. Betrachten wir uns die NewWindow-Struktur, also den Wunschzettel, in Tabelle 6 etwas genauer. Einer Komponente dieser Struktur sollten wir besondere Beachtung schenken: den IDCMP-Flags (IDCMP heißt Intuition's Direct Communication Message Ports). Hier ist die eigentliche Verbindung zum Anwender unseres Programms. Intuition teilt uns alle Aktivitäten des Anwenders über den Message-Port mit. Da uns immer nur bestimmte Nachrichten interessieren, wie zum Beispiel ein Klick auf das CloseGadget, entscheiden wir mit Hilfe dieser Flags, welche Nachrichten Intuition an das Programm senden soll. Falls das gewünschte Ereignis eintritt, setzt Intuition in der Variablen »Class« der IntuiMessa-

```
/*
\\    Max - vergrößert das aktuelle Window
*/
#include "TOOL.h"

VOID main()
{
   struct Window *win;
   struct Screen *scr;

   if(OpenLib()) puts("Kann Lib's nicht öffnen!\n");
   else{
      win=ACTIVE_WINDOW;
      scr=ACTIVE_SCREEN;
      MoveWindow(win,-(win->LeftEdge),-(win->TopEdge));
      SizeWindow(win,((scr->Width)-(win->Width)),((scr->Height)-
(win->Height)));
      CloseLib();
   }
}
```

Listing 9. Max stellt das CLI-Fenster auf maximale Größe ein

```
struct IntuiMessage
{
   struct Message ExecMessage;                    /* nur für Exec */
   ULONG Class;         /* enthält das IDCMP-Flag, z.B. MOUSEMOVE */
   USHORT Code;         /* nähere Beschreibung von Class, z.B.
SELECTDOWN */
   USHORT Qualifier;/* Tastenkombinationen (SHIFT,CTRL,ALT,AMIGA,
REPEAT) */
   APTR IAddress;/* die Adresse des Nachrichten-
Objekts, z.B. ein Gadget */
   SHORT MouseX, MouseY;          /* X- und Y-Koordinaten des
Mauspointers */
   ULONG Seconds, Micros;/* die Systemzeit zum Zeitpunkt des
Ereignisses */
   struct Window *IDCMPWindow;    /* für dieses Window ist die
Nachricht */
   struct IntuiMessage *SpecialLink;/* Verbindung zu anderen
Nachrichten */
};
```

Listing 10. Beim Auslesen des IDCMP erhalten wir einen Zeiger auf die IntuiMessage-Struktur

Die NewWindow-Struktur

LeftEdge: beschreibt die Position der linken Fensterseite relativ zum Screen. Ein Window muß sich vollständig auf einem Screen befinden, kein Teil darf »überstehen«. Daher kommen für LeftEdge nur positive Werte in Frage. Bitte Vorsicht: Intuition prüft die Werte der NewWindow-Struktur nicht!

TopEdge: ist der Abstand in Pixel, gemessen von der oberen Kante des Screens, auf dem sich das Window befindet. Alle Windows sind einem bestimmten Screen zugeordnet. Unter anderem ist zum Beispiel die Auflösung eines Windows von diesem Screen abhängig. Ein Wechsel auf einen anderen Screen ist daher nicht so einfach möglich.

Width: hier erwartet Intuition die Breite des Windows, nicht etwa die Position des rechten Fensterrandes auf dem Screen.

Height: der Wert, der die Höhe des neuen Fensters beschreibt. Die bisher besprochenen Werte beschreiben die Lage und die Ausmaße eines Intuition-Windows. Bild 2 verdeutlicht, wie die Variablen zu verstehen sind.

DetailPen: ist der Wert des Farbregisters, welches die Farbe enthält, in der zum Beispiel der Text der Titelleiste ausgegeben wird. Bei der Eingabe von -1 übernimmt Intuition die Registerangabe des Screens.

BlockPen: ein weiterer Stift. Dieser ist zum Beispiel für den Balken der Titelleiste zuständig. Siehe auch DetailPen.

IDCMPFlags: Intuition kann uns über einen Messageport mitteilen, was der Anwender unseres Programms gerade anstellt.

Flags: Diese Flags können unter anderem das Aussehen eines Windows bestimmen. Wie auch die IDCMPFlags dürfen sie, soweit sinnvoll, verknüpft werden. Tabelle 7 beinhaltet und erläutert alle Windowflags.

FirstGadget: diese Variable der NewWindow-Struktur enthält einen Zeiger auf das erste selbstdefinierte Gadget in diesem Window (NULL bei einem Window ohne Gadgets).

CheckMark: verlangt einen Zeiger auf eine Image-Struktur, die ein eigenes Menühäkchen enthält oder NULL, wenn das »Systemhäkchen« ausreicht.

Title: die Adresse des Fenstertitels.

Screen: ein Zeiger auf die Screen-Struktur oder NULL, wenn sich das Fenster auf dem Workbenchscreen befinden soll.

BitMap: die Adresse der Bitmap eines SuperBitMap-Windows.

MinWidth: enthält die kleinste Breite, auf die der Anwender das Fenster ziehen kann. Trägt man 0 ein, so ist Width die kleinste Breite.

MinHeight: bezieht sich auf die Höhe des Fensters, ansonsten wie MinWidth.

MaxWidth: enthält die größte Breite, auf die der Anwender das Fenster ziehen kann. Intuition nimmt als die größte Breite Width an, wenn man 0 einträgt.

MaxHeight: bezieht sich auf die Höhe des Fensters, ansonsten wie MaxWidth.

Type: hier verlangt Intuition noch einmal den Screentyp.

Tabelle 6. »Wunschzettel« NewWindow unter der Lupe

ge-Struktur (Listing 10) das entsprechende Flag.

Hier die möglichen IDCMP-Flags und ihre Bedeutung, in der gleichen Reihenfolge wie sie im Headerfile »intuition/intuition.h« definiert sind:

»SIZEVERIFY« Wenn dieses Flag gesetzt ist, gibt Intuition ein Signal, wenn der Benutzer versucht, die Größe des Windows zu verändern. Intuition wartet dann, bis das Programm das Signal bestätigt. In der Zwischenzeit kann beispielsweise eine Grafik fertig aufgebaut oder der neuen Größe angepaßt werden.

»NEWSIZE« Das Programm erhält eine Nachricht, wenn die Größe des Fensters verändert wurde.

»REFRESHWINDOW« Intuition informiert uns, sobald der Fensterinhalt, beispielsweise durch ein Pull-Down-Menü, zerstört wurde. Dies ist wichtig, wenn es sich um ein Fenster vom Refreshtyp »SIMPLE__REFRESH« (siehe unten) handelt. Das Programm muß dann das Fenster selbst neu aufbauen.

»MOUSEBUTTONS« Der Anwender hat eine Maustaste betätigt. Intuition hält in der Variablen »Code« der IntuiMessage-Struktur den Zustand der Tasten genau fest.

»SELECTUP« und »SELECTDOWN« bedeuten, daß die linke Maustaste gedrückt, beziehungsweise losgelassen wurde. Hat man das Windowflag »RMBTRAP« gesetzt, zeigt Intuition auch die Betätigung der rechten Maustaste an. Code unterscheidet in diesem Fall zwischen »MENUUP« und »MENUDOWN«.

»MOUSEMOVE« Intuition aktualisiert in der IntuiMessage-Struktur die X- und Y-Position des Mauszeigers bezüglich der linken oberen Ecke des Windows. Hat man das Window-Flag »REPORTMOUSE« gesetzt, erhält man bei jeder Mausbewegung die Nachricht »MOUSEMOVE« und kann die entsprechenden Koordinaten ablesen. Wie wir später noch sehen werden, ist dieses Flag auch sehr sinnvoll bei der Verwendung von Gadgets. Intuition sendet dann die Position des Mauszeigers, bezogen auf die linke obere Ecke des Gadgets. Die Verwendung des MOUSEMOVE-Flags ist etwas problematisch, weil die Nachrichten gepuffert werden. Bewegt der Anwender den Mauszeiger rasch über den ganzen Bildschirm, so strömen eine Unmenge von MOUSEMOVE-Messages auf das Programm ein – und jede muß bearbeitet und anschließend mit »ReplyMsg()« beantwortet werden.

»GADGETDOWN« und »GADGETUP« senden ein Signal, wenn über einem Gadget die linke Maustaste gedrückt beziehungsweise losgelassen wurde.

»REQSET« informiert uns über das Öffnen eines Requester auf dem Window.

»MENUPICK« Die rechte Maustaste wurde betätigt. Die Nummer des gewählten Menü-Items findet man in der IntuiMessage-Komponente Code. Falls kein Item gewählt wurde, enthält Code die Konstante »MENUNULL«.

»CLOSEWINDOW« Über dieses Flag teilt uns Intuition die Anwahl des CloseGadgets mit.

»RAWKEY« weist darauf hin, daß die Tastatur betätigt wurde. Die Variable Code der IntuiMessage-Struktur enthält die Codes der vom Benutzer betätigten Tasten. Die Variable Qualifier der gleichen Struktur beinhaltet Informationen über Tastenkombinationen mit Shift oder anderen Funktionstasten. In Bild 3 finden Sie alle RAWKEY-Codes der Amiga-Tastatur.

RAWKEY: Tastatur pur

Beachten Sie, daß sich die Codes der Zifferntasten von denen des Ziffernblocks unterscheiden. Dadurch können diese Tasten separat angesprochen werden.

»REQVERIFY« Intuition sendet diese Nachricht, sobald ein Systemrequester geöffnet werden soll und wartet, bis das Programm diese Nachricht bestätigt (siehe auch SIZEVERIFY).

»REQCLEAR«: ein Requester wurde geschlossen.

»MENUVERIFY« Intuition möchte ein Pull-Down-Menü aufklappen, sendet aber vorher diese Nachricht und wartet, bis das Programm sie bestätigt (siehe auch die anderen VERIFY-Flags).

Eine Besonderheit ist zu beachten: Wenn ein Window dieses IDCMP-Flag gesetzt hat, so empfangen alle Windows auf dem gleichen Screen die Nachricht MENUVERIFY, bevor das Menü aufklappt. Ist das Window mit gesetztem MENUVERIFY-Flag aktiv, enthält die Variable Code der IntuiMessage-

ESC 45 | F1 50 | F2 51 | F3 52 | F4 53 | F5 54 | F6 55 | F7 56 | F8 57 | F9 58 | F10 59 | DEL 46
~ \` 00 | ! 1 01 | @ 2 02 | # 3 03 | $ 4 04 | % 5 05 | ^ 6 06 | & 7 07 | * 8 08 | (9 09 |) 0 0A | _ - 0B | + = 0C | | \ 0D | BACK SPACE 41
TAB 42 | Q 10 | W 11 | E 12 | R 13 | T 14 | Y 15 | U 16 | I 17 | O 18 | P 19 | { [1A | }] 1B | 44 | HELP 5F
CTRL 63 | CAPS LOCK 62 | A 20 | S 21 | D 22 | F 23 | G 24 | H 25 | J 26 | K 27 | L 28 | : ; 29 | " ' 2A | RETURN | ↑ 4C
SHIFT 60 | 30 | Z 31 | X 32 | C 33 | V 34 | B 35 | N 36 | M 37 | < , 38 | > . 39 | ? / 3A | SHIFT 61 | ← 4F | → 4E
ALT 64 | A 66 | 40 | A 67 | ALT 65 | ↓ 4D
7 3D | 8 3E | 9 3F
4 2D | 5 2E | 6 2F
1 1D | 2 1E | 3 1F
0 0F | . 3C
- 4A | ENTER 43

Bild 3. Diese Hex-Werte liefert die IntuiMessage-Strukturkomponente Code bei einer RAWKEY-Message für die einzelnen Tasten

```
/*
\\    KMaus.c - CLI-Kommando per Mausklick. Aufrufbeispiel:
          RUN KMaus INFO
*/
#include "TOOL.h"

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung) printf("%s\n",meldung);
    CloseLib();
    exit(0);
}

VOID main(argc,argv)
    SHORT argc;
    STRPTR argv[];
{
    struct Window *win;
    ULONG class;
    USHORT code=0;

    if(argc!=2)   error("Kein CLI-Kommando?");
    if(OpenLib()) error("Kann Lib's nicht öffnen!");
    win=ACTIVE_WINDOW;              /* das gerade aktive CLI-Window */
    win->Flags|=RMBTRAP;   /* Klick auf rechte Maustaste abfangen */
    ModifyIDCMP(win,MOUSEBUTTONS);      /* IDCMP-Flags verändern */
    while(code!=MENUUP){              /* warten auf rechte Maustaste */
        Wait(1<<win->UserPort->mp_SigBit);       /* warten
        auf Message */
        class=GetMessage(win,&code,0,0);
        if((class==MOUSEBUTTONS)&&(code==SELECTUP)) Execute
        (argv[1],0,0);
    }
    win->Flags^=RMBTRAP; /* Urzustand der Window-Flags herstellen */
    ModifyIDCMP(win,0);   /* IDCMP-Flags des
    CLI-Windows zurücksetzen */
    error(0);
}
```

Listing 11. Nach dem Start von KMaus startet ein beliebiges CLI-Kommando per Mausklick. Bitte mit »RUN KMaus 'Kommando'« aufrufen.

Struktur den vordefinierten Wert »MENUHOT«, ansonsten »MENUWAITING«.

Im ersten Fall können Sie das Aufklappen des Menüstreifens ganz verbieten! Belegen Sie hierzu die Variable Code mit MENUCANCEL, bevor Sie die Nachricht beantworten.

»NEWPREFS« Der Anwender hat ein oder mehrere Werte in Preferences geändert.

»DISKINSERTED« und »DISKREMOVED« Eine Diskette wurde eingelegt beziehungsweise entfernt.

»ACTIVEWINDOW« und »INACTIVEWINDOW« Informiert, wenn ein Fenster aktiviert beziehungsweise nicht aktiviert ist.

»DELTAMOVE« Dieses Flag arbeitet mit »MOUSEMOVE« zusammen. Während MOUSEMOVE die Position des Mauszeigers bezüglich der linken oberen Fensterecke anzeigt, informiert die Kombination der beiden Flags über die Differenz (Delta) zur letzten Mausposition.

»VANILLAKEY« arbeitet wie RAWKEY, übersetzt jedoch die Keycodes in ASCII-Zeichen.

»INTUITICKS« Solange das Fenster aktiv ist, sendet Intuition laufend dieses Signal, so daß man es als einfachen Timer einsetzen kann. Diese Nachricht wird nicht gepuffert! Intuition sendet erst weiter, wenn ein INTUITICK beantwortet ist.

Wie empfängt man nun eine Nachricht von Intuition? Wie kommt man an die Adresse der IntuiMessage-Struktur? Kein Problem, der Betriebssystemteil Exec hält passende Funktionen für uns bereit. Die wichtigste davon ist ohne Zweifel:

```
GetMsg(win->UserPort);
```

Der Parameter ist der Message-Port, aus dem die Nachricht empfangen werden soll, im Beispiel der UserPort des Windows mit der Adresse »win«. Die Funktion liefert einen Zeiger auf die IntuiMessage-Struktur. Der Aufruf von ReplyMsg(&Message) beantwortet die Nachricht. Der von GetMsg() reservierte Speicherplatz für die Struktur wird wieder freigegeben.

Wir müssen demnach die Struktur **vor** der Beantwortung der Nachricht auslesen und zwischenspeichern.

Das Empfangen, Auslesen und Beantworten einer Message haben wir in Tool.h in einer Funktion realisiert. Deren Aufruf gestaltet sich denkbar einfach:

```
struct Window *w;
ULONG class;
USHORT c,i,q;
...
class=GetMessage(&w,&c,
&i,&q);
```

GetMessage() liest den Messageport aus und liefert die Komponente Class der Intui-Message-Struktur. Die Funktion erwartet vier Parameter:

»w« ist der Zeiger auf die Window-Struktur des betreffenden Windows. Die Variablen c,i und q werden in GetMessage() mit Werten belegt, weshalb wir beim Funktionsaufruf nur die Adressen übergeben dürfen.

Die erste mit dem Wert der Message-Variablen Code und »q« mit dem von Qualifier. Die Variable »i« erhält beim Arbeiten mit Gadgets die Gadget-ID.

Wenn Sie einen der aufgeführten Werte in Ihrem Programm nicht benötigen, geben Sie anstelle der Adresse einfach 0 an. Doch wie funktioniert GetMessage()? Nach der Vereinbarung der notwendigen Variablen, liest GetMsg() den UserPort des angegebenen Windows aus.

Liegt keine Nachricht an, so liefert die Funktion 0 und es erfolgt der Rücksprung. Im anderen Fall lesen wir die wichtigsten Komponenten der Nachricht aus, bevor wir sie beantworten. Die Funktion kehrt mit Class als Returnwert zurück. Ein Aufrufbeispiel finden Sie in Listing 11.

Das Programm »KMaus« erwartet beim Start einen Kommandoparameter. Nach einem Klick mit der rechten Maustaste, führt das Programm diesen Parameter als CLI-Kommando aus. Das Programmende erfolgt nach der Betätigung der rechten Taste. Das Listing ist gut dokumentiert, so daß einige wenige Worte zur Erklärung ausreichen dürften:

Nach Prüfung des Kommandoparameters und Öffnen der Libraries verändern wir in main() zwei Parameter des gerade aktiven CLI-Windows. RMBTRAP als Windowflag bewirkt, daß wir auch Nachrichten über die rechte Maustaste empfangen können.

Die IDCMP-Flags setzen wir mit der Funktion ModifyIDCMP() neu. Diese Funktion verlangt zwei Parameter: ein Zeiger auf ein Window und die neuen IDCMP-Flags. Die erste Funktion im Rumpf der folgenden Endlosschleife ist Wait(). Anstatt in einer Schleife ständig den Messageport auszulesen, warten wir mit dieser Exec-Funktion, bis tatsächlich eine Nachricht bereitsteht. Wait() hat den Vorteil, daß nicht das System, sondern nur das Programm in einen Wartezustand versetzt wird. Wir warten also, ohne das System zu belasten, auf eine Nachricht. Bis diese eintrifft, kann der Anwender problemlos im CLI (oder einem anderen Programm) weiterarbeiten.

Nachdem Sie nun wissen, was alles auf den »Wunschzettel« gehört, liegt es an Ihnen, diesen auszufüllen. Sie könnten dies mit einer statischen Initialisierung angehen oder innerhalb einer Funktion jeder Komponenten der NewWindow-Struktur einen Wert zuweisen.

Wir schlagen Ihnen eine dritte Möglichkeit vor. Nutzen Sie Tool.h und öffnen Sie in Zukunft ein Window mit einem einzigen Aufruf:

```
struct Window *w;
struct Screen *s;
SHORT x,y,w,h;
ULONG i,f;
STRPTR t;
struct Gadget *g;
...
w=GetWindow(s,x,y,w,h,
i,f,t,g);
```

Und jetzt: Fenster auf

Die Parameter der Funktion GetWindow() sind: s – die Adresse des Screens, auf dem das Window erscheinen soll. Wünschen Sie den gerade aktiven Screen, geben Sie einfach NULL an.

x und y – die Position der linken oberen Ecke des Windows auf dem Screen.

b und h – Breite und Höhe des Windows.

i – die IDCMP-Flags.

w – die Windowflags. (siehe Tabelle 7)

t – die Adresse des Window-Titels und

g – der Zeiger auf das erste selbstdefinierte Gadget im Window (falls gewünscht).

Entsprechend diesen Parametern initialisiert GetWindow() die meisten Werte der NewWindow-Struktur. Die anderen legen wir fest, um den Aufruf unserer Funktion nicht unnötig zu komplizieren. Die Intuition-Funktion OpenWindow() öffnet das gewünschte Fenster. Als Parameter geben wir die Adresse der NewWindow-Struktur an. GetWindow() reicht den von

```
/*
\\    Demo.c - demonstriert Aufruf von GetWindow()
*/
#include "TOOL.h"

struct Window *win;

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung) printf("Kann %s nicht öffnen!\n",meldung);
    if(win)     CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    ULONG class;

    if(OpenLib()) error("Lib's");
    win=GetWindow(NULL,10,20,200,100,INACTIVEWINDOW,ACTIVATE,"Demo",
    NULL);
    if(win==NULL) error("Window");
    while(class!=INACTIVEWINDOW){ /* warten bis Window inaktiviert
    ist */
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage(win,0,0,0);
    }
    error(0);
}
```

Listing 12. Ein einfaches Beispiel – so wird GetWindow() aufgerufen

OpenWindow() gelieferten Zeiger an die Window-Struktur des neuen Fensters weiter. Konnte OpenWindow() das Fenster nicht öffnen, erhalten wir den Wert 0.

Listing 12 ist ein einfaches Beispiel für den Aufruf der neuen Funktion. Schauen Sie es sich doch einmal genauer an. Interessant ist, neben dem Aufruf der neuen Funktion, wie das Programmende ausgelöst wird - wissen Sie's?

Die von GetMessage() gelieferte Window-Struktur enthält jede Menge wichtiger Informationen. Listing 13 zeigt das dokumentierte Listing dieser Struktur. Wenn noch Fragen auftauchen - einfach ausprobieren.

Wie für die Screens beinhaltet Intuition auch für Windows einige für den Programmierer interessante Funktionen. Diese erwarten allesamt die Adresse der Window-Struktur. Wir nennen Sie kurz »win«:

ActivateWindow(win) aktiviert ein inaktives Fenster.

CloseWindow(win) schließt ein Fenster.

MoveWindow(win,xp,yp) bewegt das Window um xp Pixel nach rechts und um yp Pixel nach unten. Negative Werte bewirken eine Bewegung in der entgegengesetzten Richtung (siehe Listing 9).

RefreshWindowFrame(win) der Windowrahmen wird erneuert.

ReportMouse(win,on): setzt das REPORTMOUSE-Flag (bei on gleich 1) oder löscht es (bei on gleich 0).

SetWindowTitle(win,&WTitle,&STitle): verändert Screen- und Fenstertitel. Parameter 2 und 3 sind die Adressen der neuen Titel für Fenster und Screen. Bei der Angabe von -1, ändert sich der Titeltext nicht.

SizeWindow(win,x,y): vergrößert das Fenster um x Punkte nach rechts und y Punkte nach unten (Listing 9). Negative Werte verkleinern das Fenster entsprechend. Achtung: Intuition prüft die Werte nicht. Wenn Sie also einen Wert angeben, der Ihr Fenster beispielsweise über den Screenrand ragen läßt, so wird Ihnen das der Guru übelnehmen.

ViewAddress(win): liefert die Adresse der ViewPort-Struktur. Die Adresse der View-Struktur erhalten Sie ganz ähnlich mit ViewAddress().

WindowLimits (win,MinX,MinY,MaxX,MaxY): verändert die in der NewWindow-Struktur vereinbarten Limits. Hatte die Funktion Erfolg, liefert sie 1, ansonsten 0.

WindowToBack(win) und **WindowToFront(win)**: verrichten die Arbeit der Window-Depth-Gadgets.

Das waren viele Routinen in geballter Form. Testen Sie sie ausführlich, damit Sie von den Möglichkeiten, die Ihnen zur Verfügung stehen, einen Eindruck gewinnen.

Hier noch einige Probleme, an denen Sie sich selbst testen können:

- Öffnen Sie ein Window mit allen Systemgadgets. Programmende soll bei der Anwahl des CloseGadgets erfolgen.
- Schreiben Sie ein Programm, das den beim Aufruf mitgegebenen Kommandoparameter als Windowtitel ins aktuelle CLI-Window einträgt. Der Screentitel soll dabei nicht verändert werden.
- Öffnen Sie ein kleines Window, das Ihnen die Möglichkeit gibt, »RAWKEY«-Messages zu empfangen. Geben Sie die Key-Codes und Key-Qualifiers auf das aktuelle CLI-Window aus. Beobachten Sie, wie sich der Wert Qualifier beim Niederhalten einer Taste verändert (Repeat-Funktion).

Nachdem Sie die grundlegenden Intuition-Elemente Screen und Window beherrschen, wollen wir unseren Spaziergang durch den Betriebssystemteil Intuition unterbrechen und uns, in einem Exkurs, den Grafikfunktionen zuwenden.

Die Window-Flags

WINDOWSIZING: das neue Fenster verfügt über das Systemgadget, mit dessen Hilfe der Anwender die Größe des Fensters festlegen kann.

SIZEBRIGHT: das WINDOWSIZING-Gadget ist am rechten Rand angebracht. Dies ist die Voreinstellung. Der Nachteil: der ganze rechte Rand des Fensters kann in der Breite des Gadgets nicht beschrieben werden.

SIZEBBOTTOM: das WINDOWSIZING-Gadget ist an der rechten Seite des unteren Fensterrandes angebracht. Man kann nun volle 80 Zeichen auf dem Fenster darstellen. Allerdings kann man in dieser Einstellung den unteren Fensterrand nicht beschreiben.

WINDOWDRAG: die Titelleiste verwandelt sich in ein Verschiebe-Gadget.

WINDOWDEPTH: Intuition montiert die Depth-Gadgets.

WINDOWCLOSE: ist das Gadget zum Schließen eines Fensters.

SMART_REFRESH: Intuition kopiert Fensterteile, die durch Überlagerung anderer Elemente verlorengehen, in einen reservierten Speicher, um das Fenster später wieder regenerieren zu können. Bereiche, die durch Verkleinern des Fensters verlorengehen, kann Intuition nicht wiederherstellen.

SIMPLE_REFRESH: Intuition teilt uns über IDCMP mit, wenn das Fenster renovierungsbedürftig ist. Das Programm muß dann selbst den Fensterinhalt erneuern. Dies geschieht auf folgende Art:

```
BeginnRefresh(&window);
NeuAufbau();
EndRefresh(&window,ok);
```

Nach dem Aufruf der Funktion BeginRefresh() reagiert Intuition nur noch auf Funktionen, die sich auf den sichtbaren Bereich des Fensters beziehen. Es folgt nun eine Routine zum Fensteraufbau. Danach schalten wir Intuition mit EndRefresh() wieder in den Normalzustand zurück. Die Parameter der beiden Refreshfunktionen sind Zeiger auf das entsprechene Window. Der zusätzliche Parameter »ok« von EndRefresh() ist TRUE, wenn das Fenster komplett erneuert wurde und FALSE, wenn nicht.

SIMPLE-REFRESH: ist zwar etwas umständlich zu bedienen, spart jedoch gegenüber SMART_REFRESH oder SUPER_BITMAP enorm an Speicherplatz.

SUPER_BITMAP: ist ein weiterer Window-Refreshtyp. Der komplette Fensterinhalt befindet sich als Kopie im Speicher. Möglicherweise stellt das Fenster nur einen Ausschnitt des gesamten Bereichs dar. Diese Art des Refreshing ist sehr speicherintensiv und findet seltener eine Anwendung, weshalb wir sie auch in Tool.h nicht berücksichtigen.

BACKDROP: das Window befindet sich immer hinter allen anderen Windows.

REPORTMOUSE: die Koordinaten des Mauszeigers werden bei jeder Änderung über IDCMP aktualisiert.

GIMMEZEROZERO: bei diesem Window zeichnet Intuition den Rahmen, die Titelleiste und die Systemgadgets getrennt vom eigentlichen Fensterinhalt. Sie sind dadurch vor Überschreiben geschützt.

BORDERLESS: ein Fenster ohne Rahmen wird geöffnet.

ACTIVATE: das Fenster ist gleich nach dem Öffnen aktiviert.

RMBTRAP: in diesem Window verhält sich die rechte Maustaste wie die linke.

NOCAREREFRESH: Intuition erneuert das Window nicht und schickt uns auch keine entsprechende Nachricht.

Tabelle 7. Die Window-Flags auf einen Blick



Auf diese Weise kommt nicht zu viel Therorie auf einmal auf Sie zu. Sie können viele Funktionen testen und dabei Ihr Wissen festigen.

Anders als bei vielen anderen Computern sind im Betriebssystem des Amiga eine große Anzahl, zum Teil sehr leistungsfähiger Grafikroutinen enthalten. Dabei fällt dem C-Programmierer die einfache Handhabung dieser Routinen angenehm auf.

Für die Verwendung dieser Routinen ist die RastPort-Struktur sehr wichtig. Machen wir uns gleich zu Beginn mit ihr vertraut (siehe Listing 14).

```
struct Window
{
    struct Window *NextWindow;/* Adresse des nächsten Windows auf
    dem Screen */
    SHORT LeftEdge, TopEdge;        /* Position der linken oberen
    Fensterecke */
    SHORT Width, Height;                    /* Breite und Höhe des
    Windows */
    SHORT MouseY, MouseX;  /* Mauszeiger-Koordinaten, bezogen
    auf das Window */
    SHORT MinWidth, MinHeight;                /* Minimalwerte des
    Fensters */
    USHORT MaxWidth, MaxHeight;               /* Maximalwerte des
    Fensters */
    ULONG Flags;                              /* die Windowflags */
    struct Menu *MenuStrip;          /* Zeiger auf den Menü-Header */
    UBYTE *Title;                         /* der Titel des Windows */
    struct Requester *FirstRequest;   /* Zeiger auf den 1. aktiven
    Requester */
    struct Requester *DMRequest;        /* Zeiger auf den
    DoubleMenu-Requester */
    SHORT ReqCount;   /* Anzahl der zum Window gehörenden offenen
    Requester */
    struct Screen *WScreen;  /* der Screen, auf dem sich das Window
    befindet */
    struct RastPort *RPort;        /* der Zeiger auf den RastPort des
    Windows */
    BYTE BorderLeft, BorderTop, BorderRight, BorderBottom;     /*
    Rahmendaten */
    struct RastPort *BorderRPort;               /* RastPort des Rahmens */
    struct Gadget *FirstGadget;     /* Zeiger auf das erste Gadget */
    struct Window *Parent, *Descendant; /* vorheriges und
    nachfolgendes Win. */
    USHORT *Pointer;                  /* Zeiger auf die Daten
   des Mauszeigers */
    BYTE PtrHeight,PtrWidth;                  /* Höhe und Breite des
    Zeigers */
    BYTE XOffset, YOffset;          /* Lage des >>heißen Punktes<
    < im Pointer */
    ULONG IDCMPFlags;               /* IDCMP-Flags aus der NewWindow-
    Struktur */
    struct MsgPort *UserPort, *WindowPort;       /* Message-Ports des
    Windows */
    struct IntuiMessage *MessageKey;          /* Adresse der Message
    Struktur */
    UBYTE DetailPen, BlockPen;        /* Farben, wie in NewWindow
    festgelegt */
    struct Image *CheckMark;/* Zeiger auf die Daten eines
    Alternativhäkchens */
    UBYTE *ScreenTitle;                  /* Titeltext des Screens */
    SHORT GZZMouseX,GZZMouseY;     /* Pointer-Koordinaten in einem
    GZZ-Window */
    SHORT GZZWidth,GZZHeight; /* Breite und Höhe eines GimmeZeroZero-
    Windows */
    UBYTE *ExtData;            /* Verbindung zu externen Datenblöcken */
    BYTE *UserData;  /* Verbindung zu userdefinierten Datenblöcken */
    struct Layer *WLayer;  /* Adresse der Layer-
Struktur (s.a. RPort->     Layer) */
    struct TextFont *IFont;                        /* Window-Font */
};
```

Listing 13. Die Window-Struktur enthält jede Menge Informationen

Die Aufgabe dieses Datenblocks erkennen Sie am besten, wenn Sie sich über die Aufgaben der einzelnen Komponenten im klaren sind:

`struct Layer *Layer`

ist die Adresse der Layer-Struktur des Elementes, dem der RastPort angehört. Layers sind grundlegende Grafikelemente,

Das mag Sie an die Eigenschaften eines Windows erinnern, tatsächlich ist es so, daß Windows nichts anderes als erweiterte Layers sind.

`struct BitMap *BitMap`

die BitMap-Struktur (Listing 15) ist ein Bestandteil der Screen Struktur.

`USHORT *AreaPtrn`

ein Zeiger auf ein Array, welches die Daten des aktuellen Füllmusters enthält. Jede Funktion, die in diesem RastPort Flächen füllt, etwa RectFill(), Flood() etc., füllt mit diesem Muster.

`struct TmpRas *TmpRas`

ist ein Zwischenspeicher für Füllfunktionen. Das (temporäre) Raster muß mindestens so groß sein, wie die zu füllende Fläche.

`struct AreaInfo *AreaInfo`

Adresse der AreaInfo-Struktur, welche zum Beispiel die Eckpunkte eines Area-Polygons enthält.

`struct GelsInfo *GelsInfo`

Zeiger auf eine Struktur, die für Grafikelemente wie zum Beispiel BOBs (Blitter-Objects) im RastPort wichtig ist.

`UBYTE Mask`

mit dieser Maske können wir festlegen welche Plane vor Überschreiben geschützt werden soll. Eine geschützte Plane wird durch eine 0 dargestellt. Wir lernen später diesen Wert zu verändern.

`BYTE FgPen,BgPen,AOlPen`

die Farben für Zeichen-, Hintergrund-, Outlinestift, auch APen, BPen und OPen genannt.

`BYTE DrawMode`

der aktuelle Zeichenmodus. Es kommen »JAM1«, »JAM2«, »COMPLEMENT«, »INVERSEVID« und Kombinationen davon in Frage.

`BYTE AreaPtSz`

ist die Höhe des Füllmusters als Potenz zur Basis 2. Die folgenden drei Komponenten sind für die interne Nutzung des Betriebssystems bestimmt. Wir besprechen weiter ab:

```
struct RastPort
{
    struct Layer *Layer;
    struct BitMap *BitMap;
    USHORT *AreaPtrn;
    struct TmpRas *TmpRas;
    struct AreaInfo *AreaInfo;
    struct GelsInfo *GelsInfo;
    UBYTE Mask;
    BYTE FgPen,BgPen,AOlPen;
    BYTE DrawMode;
    BYTE AreaPtSz;
    BYTE linpatcnt;
    BYTE dummy;
    USHORT Flags;
    USHORT LinePtrn;
    SHORT cp_x, cp_y;
    UBYTE minterms[8];
    SHORT PenWidth;
    SHORT PenHeight;
    struct TextFont *Font;
    UBYTE AlgoStyle;
    UBYTE TxFlags;
    UWORD TxHeight,TxWidth,
TxBaseline;
    WORD TxSpacing;
    APTR *RP_User;
    ULONG longreserved[2];
    UWORD wordreserved[7];
    UBYTE reserved[8];
};
```

Listing 14. Die RastPort-Struktur beschreibt die Zeichenfläche

`USHORT LinePtrn`

dieses Zeichenmuster für Linien verändern wir mit SetDrPt().

`SHORT cp_x, cp_y`

die Koordinaten des (unsichtbaren) Zeichencursors. Die Tool.h-Funktion GetPos() liest diese Werte. Die Bedeutung des Arrays minterms und die der Variablen PenWidth und PenHeight ist uns unbekannt. In der vorliegenden Betriebssystemversion beträgt jedenfalls Breite und Höhe des Zeichenstifts immer nur 1 Pixel.

`struct TextFont *Font`

Zeiger auf TextFontStruktur des im RastPort aktiven Zeichensatzes.

`UBYTE AlgoStyle`

die aktuelle SoftStyle-Schriftart, also unterstrichen, kursiv oder fett.

`UBYTE TxFlags`

Näheres über den Zeichensatz, beispielsweise ob es sich um einen DiskFont oder einen ROM-Font handelt.

`UWORD TxHeight`

die Höhe der Zeichen des aktuellen Zeichensatzes.

`UWORD TxWidth`

die Breite eines einzelnen Zeichens im Zeichensatz.

```
UWORD TxBaseline
```

die Position der Basislinie der Zeichen, ist identisch mit der Y-Position des Zeichencursors bei der Textausgabe.

```
WORD TxSpacing
```

der Abstand zwischen zwei Zeichen in Pixel (normal 0).

Die restlichen Daten sind für das System reserviert. Sie sehen, RastPort ist eine sehr umfangreiche Struktur, die interessante Daten für den Programmierer bereithält. Viele Elemente darin ändern sich mit dem Aufruf bestimmter Grafikfunktionen. Diese Funktionen – wir werden einige davon testen – verlangen fast alle die Adresse der RastPort-Struktur. Wir beziehen diese Adresse für unser Window aus der Window-Struktur oder für einen Screen aus der Screen-Struktur. Beispiel:

```
struct RastPort *RP;
...
RP=win->RPort;
```

Zum Zeichnen braucht man immer einige Zeichenstifte – auch auf einem Computer. Dem Amiga-Programmierer stehen drei verschiedene Stifte zur Auswahl. Zum Zeichnen von Linien oder Punkten benutzen wir »APen«. Sein Kollege »BPen« ist für die Hintergrundfarbe zuständig, zum Beispiel bei der Textausgabe. »OPen« ist der dritte im Bunde. Er zeichnet, sobald man eine Farbe festgelegt hat, bei Funktionen wie »RectFill()« einen Rahmen. Diese Darstellung schaltet man mit dem Makro »BNDRY-OFF(RP)« aus. Durch Aufruf des Makros »SetOPen()« setzen wir in der angegebenen RastPort-Struktur das Flag »AREAOULINE« und durch »BNDRYOFF()« wird es wieder gelöscht.

Zeichenstifte für alle Fälle

Die Farbe eines Stiftes wählt man mit der Set-Funktion. Ein Beispiel für APen:

```
SetAPen(RP, FR);
```

Die Parameter sind RastPort-Adresse und das Farbregister, welches die gewünschte enthält.

Es sind beim Zeichnen verschiedene Zeichenmodi möglich, man stellt sie mittels SetDrMd(RP,modus) ein. Die Modi:

JAM1 - APen zeichnet in der angewiesenen Farbe;
JAM2 - BPen zeichnet zum Beispiel bei Text den Hintergrund;
JAM1/COMPLEMENT und JAM2/COMPLEMENT – komplementiert die Bits unter der Cursorposition, das heißt: stammt die Farbe des Pixels unter dem Cursor aus Farbregister %10101, färbt sich der Punkt beim Überzeichnen in der Farbe, welche Farbregister %01010 enthält;
JAM1/INVERSVID und JAM2/INVERSVID - zeichnet in negativer Darstellungsform.

Die Stifte APen und BPen befinden sich beide über einem imaginären Grafikcursor. Dieser läßt sich mit der Funktion Move(RP,x,y) an jede beliebige Stelle des RastPorts bewegen. Er bewegt sich dabei »über« der Zeichenfläche und hinterläßt daher keine sichtbaren Spuren.

```
/*
\\    Lupe.c - vergrößert die Pixel unter dem Pointer
*/
#include "TOOL.h"

#define RP_S (&ACTIVE_SCREEN->RastPort)
#define RP_W (win->RPort)
#define XPos (ACTIVE_SCREEN->MouseX)
#define YPos (ACTIVE_SCREEN->MouseY)

struct Window *win;

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung)  printf("Kann %s nicht öffnen!",meldung);
    if(win)      CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    ULONG idcmp=CLOSEWINDOW| NEWSIZE| MOUSEMOVE,
          flags=REPORTMOUSE| WINDOWDRAG| WINDOWSIZING|
                SIZEBBOTTOM| WINDOWCLOSE| ACTIVATE,
          class=0;
    SHORT x1,y1,x2,y2,
          width=21,height=8;
    COUNT i,j;

    if(OpenLib()) error("Lib's");
    if(!(win=GetWindow(NULL,0,0,110,60,idcmp,flags,
    "Gummi-Lupe",NULL)))
                error("Window");
    SetDrMd(RP_W,JAM1);
    SetOPen(RP_W,3);
    while(class!=CLOSEWINDOW){          /* Programmende über
    CloseGadget */
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage(win,0,0,0);
        if(class==MOUSEMOVE){
            for(i=-2;i<3;i++){
                for(j=-2;j<3;j++){
                    SetAPen(RP_W, ReadPixel(RP_S,XPos+j,YPos+i));
                    x1= 2+(j+2)*width;
                    y1=10+(i+2)*height;
                    x2= 2+(j+3)*width;
                    y2=10+(i+3)*height;
                    RectFill(RP_W,x1,y1,x2,y2);
                }
            }              /* eventuell gepufferte MOUSEMOVEs
                löschen: */
            while((class=GetMessage(win,0,0,0))==MOUSEMOVE);
        }
        if(class==NEWSIZE){          /* die Window-Größe wurde
                verändert */
            width =(SHORT)(win->Width - 5)/5;     /* Breite der
            Rechtecke */
            height=(SHORT)(win->Height-20)/5;      /* Höhe der
            Rechtecke */
            SetRast(RP_W, 0);               /* RastPort komplett
            löschen */
            RefreshWindowFrame(win);            /* Rahmen neu
            zeichnen */
        }
    }
    error(0);
}
```

Listing 15. Mit der Lupe auf Pixelsuche

Anders die Funktion Draw (RP,x,y). Der Stift »senkt« sich an der aktuellen Cursorposition ab und zeichnet eine Linie zu den neuen, als Parameter mitgegebenen Koordinaten x und y.

Eine geknickte Linie erhalten wir mit dem Aufruf der Funktion PolyDraw(RP,n,&koordinaten[0]). Der Parameter n ist dabei die Anzahl der Knickpunkte plus Anfangs- und Endpunkt der Linie. »koordinaten« ist ein Zeiger auf ein Array, das die Koordinaten der Punkte enthält. Dieser Aufruf zeichnet ein Dreieck:

```
USHORT koord[]={
    0,100,
  100,100,
   50, 50,
    0,100
};
funktion()
{
 PolyDraw(RP,4,&koord[0
]);
}
```

Wenn Ihnen die einfachen Linienzüge zu eintönig sind, entwerfen Sie sich ein eigenes Linienmuster. SetDrPt(RP, muster) verlangt als Muster einen 16-Bit-Wert. Bits, die in diesem »WORD« gesetzt sind, werden auch in der Linie gesetzt. Gelöschte Bits bedeuten gelöschte Punkte.

Ein einzelnes Pixel des RastPorts könnten Sie färben, indem Sie Move() und Draw() mit gleichen Parametern aufrufen. Es existiert jedoch eigens für diesen Zweck eine spezielle Funktion. WritePixel(RP,x,y) zeichnet einen Punkt an der angegebenen x/y-Position des RastPorts. Das Gegenstück dazu, ReadPixel(RP,x,y), liefert die Nummer des Farbregister aus dem die Farbe des Punktes an der gewünschten x/y-Position stammt. Kehrt die Funktion mit einem negativen Ergebnis zurück, konnte sie das Register nicht ermitteln.

WritePixel() färbt einen einzigen Punkt des RastPorts, SetRast(RP, FR) dagegen alle, und zwar in der Farbe, die das Farbregister FR enthält. Die Funktion SetWrMsk(RP, maske) schützt Bitplanes vorm Überzeichnen. »maske« ist ein 16-Bit-Wert, deren unterste 6 Bit die sechs BitPlanes repräsentieren. Bit 0 steht dabei für Plane 0. Um eine Plane zu schützen, löscht man das entsprechende Bit des Parameters maske.

Ein ausgefülltes Rechteck erhalten wir mit dem Aufruf der Funktion RectFill(RP,x1,y1,x2,

y2). Die Parameter x1 und y1 beschreiben die Position der linken oberen Ecke, x2 und y2 die der rechten unteren. Setzen Sie doch einmal vor dem Aufruf von RectFill() die Stifte APen und OPen auf verschiedene Farben. Das Rechteck erscheint dann eingerahmt auf dem Bildschirm.

Ein Beispiel dafür finden Sie in Listing 15. Nach dem Start von »Lupe« erscheinen 25 Pixel unter dem Mauszeiger vergrößert in einem Window. Auch die Funktionen SetRast() und ReadPixel() finden in diesem gerade 60 Zeilen langen Listing eine Anwendung. Nachdem in der Hauptfunktion der Zeichenmodus auf JAM1 eingestellt ist, legen wir für den OutlinePen die Farbe 4 des aktuellen Screens fest (bei einer Original-Workbench orange).

Es folgt die Hauptschleife des Programms, die erst unterbricht, wenn der Anwender das CloseGadget wählt. Bei jedem »MOUSEMOVE« zeichnen wir ein orange eingerahmtes Rechteck in der Farbe des APen. Doch woher nehmen wir die Farbinformation? Nun, ReadPixel() liefert die Farbe unter dem Pointer, warum sollten wir die Information in einer Variablen zwischenspeichern?

Also dient hier der Aufruf der Funktion ReadPixel() als aktueller Parameter der Funktion SetAPen(). Nach den verschachtelten »for«-Schleifen löschen wir eventuell anstehende »MOUSEMOVE«-Messages in einer »while«-Schleife. Wie Sie wissen, puffert sonst Intuition diese Nachrichten.

Kommunikation mit Messages

Die Variable class verläßt diese Schleife mit einem Wert, der sich von MOUSEMOVE unterscheidet. Wir müssen prüfen, ob es sich dabei um die Message »NEWSIZE« handelt. Daher darf die »if(class== NEWSIZE)«-Anweisung kein »else«-Zweig der »if(class== MOUSEMOVE)«-Anweisung sein.

Handelt es sich um eine Nachricht der Klasse NEWSIZE, so passen wir die Rechtecke der neuen Fenstergröße an. Wer statt Rechtecke eher Rundliches mag, kommt mit DrawEllipse(RP,x,y,rh,rv) auf seine Kosten. Die Parameter x und y beschreiben die Koordinaten des Ellipsenmittelpunktes. Den Radius in horizontaler Richtung gibt rh an und den in vertikaler Richtung rv. Ein aus DrawEllipse() entwickeltes Makro ist DrawCircle(RP,x,y,r). Es zeichnet einen Kreis mit dem Radius r an die Position x/y. DrawCircle() ist im Headerfile gfxmacros.h definiert. Dort findet man auch OFF__DISPLAY und ON__DISPLAY, mit deren Hilfe man die Bildschirmdarstellung ab- beziehungsweise einschalten kann.

Es stehen uns drei Funktionen zur Verfügung, die den Elektronenstrahl der Bildröhre abfragen. WaitTOF() wartet, bis sich der Strahl wieder in der ersten Zeile des Monitors befindet. WaitBOVP(&VP) wartet, bis der Strahl die unterste Zeile des angegebenen ViewPorts erreicht hat. Parameter ist die Adresse des ViewPorts. VBeamPos() schließlich ermittelt die aktuelle vertikale Position des Elektronenstrahls.

Lassen Sie uns an dieser Stelle eine kleine Pause einlegen, um die neuen Informationen verdauen zu können. Nutzen Sie die Verweilzeit, um die bisher besprochenen Routinen ausgiebig zu testen. Fassen Sie mehrere Grafikroutinen zu einer einzigen Funktion zusammen, die vielleicht eine Strichpunkt-Linie zeichnet oder einen Rahmen.

```
/*
\\  Flow.c - stellt ein Textfile in Laufschrift dar
*/
#include "TOOL.h"
#define RP win->RPort

LONG size;
UBYTE *buf;
struct Window *win;

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung) printf("Kann %s nicht öffnen!\n",meldung);
    if(win)     CloseWindow(win);
    if(buf)     FreeFile(buf,size);        /* Speicher für File
freigeben */
    CloseLib();
    exit(0);
}

VOID Scroll(c)
    UBYTE *c;
{/* schreibt ein Zeichen und scrollt das Textband acht Pixel
    nach links */
    SHORT i;

    if((*c<32)||((*c>127)&&(*c<160))) *c=' ';  /* Sonderzeichen
    löschen */
    Move(RP,631,13);
    Text(RP,c,1);                          /* Textausgabe eines
    Zeichens */
    WaitTOF();
    for(i=0;i<8;i++) ScrollRaster(RP,1,0,2,6,637,15);
    if(GetMessage(win,0,0,0)) error(0);        /* Programmabbruch */
}

VOID main(argc,argv)
    SHORT argc;
    STRPTR argv[];
{
    COUNT i;
    ULONG idcmp=MOUSEBUTTONS|RAWKEY|INACTIVEWINDOW,       /* IDCMP-
    Flags */
          flags=ACTIVATE;                          /* Window-Flags */

    if(argc!=2) printf("Aufruf: %s Filename\n",argv[0]);
    else{
        if(OpenLib()) error("Libraries");
        if(!(win=GetWindow(NULL,0,0,640,23,idcmp,flags,NULL,NULL)))
            error("Window");
        if(!(buf=GetFile(argv[1],&size))) error(argv[1]); /* File
        laden */
        SetDrMd(RP,JAM1);
        SetAPen(RP,1);
        for(i=0;i<size;i++) Scroll(&buf[i]);
        Delay(200);
        error(0);
    }
}
```

Listing 16. Eine Alternative zum CLI-Kommando »TYPE?«: Flow listet ein File als Laufschrift.

In Tool.h haben wir bereits einige eigene Grafikfunktionen definiert. Es stehen Ihnen nun die folgenden Funktionen zur Verfügung:

Line(RP,col,x1,y1,x2,y2): zeichnet in dem angegebenen RastPort eine Linie in der Farbe col, von der Position x1/y1 zur Position x2/y2.

ILine(): zeichnet ebenfalls eine Linie, aber im JAM1:COMPLEMENT-Modus. Die Angabe einer Farbe ist demnach unnötig, ansonsten sind die Parameter die gleichen wie bei Line().

Zwei Aufrufe dieser Funktion mit gleichen Parametern heben sich wegen der doppelten Komplementierung auf. So ist das sogenannte Rubberbanding, Sie kennen es vom Arbeiten (vergrößern, verkleinern, ändern der Position) mit Windows, realisiert.

Block(RP,col,x1,y1,x2,y2): zeichnet einen rechteckigen Block in der Farbe col. Die Position der linken oberen Ecke liegt bei x1/y1, die der rechten unteren Ecke bei x2/y2.

IBlock(RP,x1,y1,x2,y2): hebt einen gewünschten rechteckigen Ausschnitt durch Komplementierung der Farbregister hervor. **IRast(RP)** komplementiert sogar den ganzen RastPort. GetPos() ermittelt die aktuelle Position des Grafikcursors. Nach dem folgenden Beispielaufruf enthalten x und y die entsprechenden Koordinaten:

```
SHORT x,y;
GetPos(RP,&x,&y);
```

GetCol(&VP,nr,&r,&g,&b) haben wir schon kurz angesprochen. Die Funktion liest das mit nr verlangte Farbregister des ViewPorts mit der Adresse VP aus und belegt die Variablen r, g und b mit den RGB-Werten. GetCol() kehrt mit dem Inhalt des Farbregisters zurück. Auch hier zur Verdeutlichung ein Beispiel:

```
SHORT col,r,g,b;
col=GetCol(&VP,1,&r,&g,
    &b);
```

Nach diesem Aufruf enthält col den Inhalt den Farbregisters 1, die Variable r den Rot-Anteil, g den Grün-Anteil und b den Blau-Anteil der Farbe. Im normalen CLI-Window enthält Farbregister 1 den Wert für Weiß. Demnach hätte col den Wert 0xFFF und r, g und b jeweils den Wert 0xF.

Schauen Sie sich bitte die neuen Tool.h-Funktionen genau an und lesen Sie erst weiter, wenn Sie wirklich wissen, wie sie funktionieren.

Wir haben noch lange nicht alle Grafikroutinen untersucht. »Graphics.library« hat noch einige Überraschungen für uns bereit, zum Beispiel:

Es ist immer wieder beeindruckend, wenn ganze Grafiken oder Teile davon, in einer gleichmäßigen Bewegung auf dem Bildschirm verschoben werden. Auch hier unterstützt uns das Betriebssystem mit einer speziellen Routine: ScrollRaster(RP,r,u,x1,y1,x2,y2). Die Parameter x1/y1 und x2/y2 beschreiben, wie bei der oben beschriebenen Block()/-Funktion, ein Rechteck. ScrollRaster() verschiebt dieses Rechteck »r« Pixel nach rechts und »u« Pixel nach unten. Negative Werte bewirken ein Scrolling in die entgegengesetzte Richtung. BPen füllt den frei werdenden Bereich.

Listing 16 demonstriert eine Einsatzmöglichkeit der ScrollRaster()- Funktion. Rufen Sie »Flow« mit dem Namen eines Textfiles als Kommandoparameter auf. Das fertig übersetzte Programm listet das angegebene File als Laufschrift. Schauen wir uns das relativ kurze Listing einmal genauer an: Die Funktion main() öffnet, nach einer Überprüfung des Kommandoparameters, die Libraries und das Window. Die Tool.h-Funktion GetFile() lädt das gewünschte File. Wir befassen uns im Abschnitt AmigaDOS noch ausführlicher mit GetFile().

Die Aufrufe der Funktionen SetDrMd() und SetAPen() legen daraufhin Zeichenmodus und -farbe fest. Scroll() gibt jedes einzelne Zeichen ganz rechts im neu geöffneten Fenster aus. Sofort verschieben wir das schmale Band mit den Zeichen um die Breite eines Zeichens nach links. Das Scrolling erfolgt Pixel für Pixel, um eine möglichst ruckfreie Bewegung zu erreichen. Sobald das Programm einen Mausklick, die Betätigung einer Taste oder die Inaktivierung des Windows registriert, erfolgt das Programmende.

Experimentieren Sie doch ein wenig mit ScrollRaster() – es lohnt sich.

Eine Routine, die Farbe ins Bild bringt, ist Flood(RP,modus, x,y). Sie füllt beliebige Flächen in der Farbe des APen. Dabei unterscheiden wir zwei Modi, die wir mit dem zweiten Parameter des Funktionsaufrufes mitteilen.

Ist modus gleich 0, füllt Flood() eine Fläche, die in der Farbe des OPen umrandet ist. Die Parameter x/y stehen für eine beliebige Position innerhalb der Umrandung.

Modus gleich eins bedeutet, daß Flood() die zusammenhängende Fläche füllt, die die gleiche Farbe des Pixels unter der x/y Position hat. Rekursive Funktionen wie Flood() sind sehr speicherintensiv. Es ist daher notwendig, für kurze Zeit eine ausreichend große Bit-Ebene, ein sogenanntes Temporär-Raster, anzulegen. Listing 17 zeigt, wie Sie vorgehen können.

```
/*
\\    Flood - demonstriert Flood() und SetAfPt()
*/
#include "TOOL.h"
#define RP win->RPort

/*UWORD data[]={
   0, 0xFF, 0,    0xFF,
   0, 0,    0xFF, 0xFF
};*/

VOID main()
{
   struct Window *win;
   struct TmpRas Raster;
   PLANEPTR Bitplane;

   if(OpenLib()||(!(win=GetWindow(NULL,0,0,320,256,0,ACTIVATE,
   "Flood",0))))
      printf("Nix geht mehr!!");
   else{
      Bitplane=AllocRaster(320,256);
      InitTmpRas(&Raster,Bitplane,RASSIZE(320,256));
      RP->TmpRas=&Raster;
      SetAPen(RP,3);
      DrawCircle(RP,160,128,100);
/*    SetAfPt(RP,&data[0],-2);    */
      Flood(RP,1,160,128);
      Delay(300);
      FreeRaster(Bitplane,320,256);
      CloseWindow(win);
      CloseLib();
   }
}
```

Listing 17. Füllen mit Komfort – die Flood() Funktion. Ruf' doch mal auf.

Zu Beginn, nach den üblichen Schritten, reserviert AllocRaster() den notwendigen Speicherplatz. Die Verwaltung unseres Temporär-Rasters übernimmt die Struktur TmpRas, deren Adresse wir an der entsprechenden Stelle innerhalb der RastPort-Struktur hinterlegen müssen.

InitTmpRas() initialisiert uns das Raster an der als ersten Parameter angegebenen Adresse. Wir tragen sie in die RastPort-Strukturkomponente TmpRas ein. FreeRaster() gibt den belegten Speicher wieder frei.

In der abgedruckten Form färbt Listing 18 eine Fläche einheitlich ein. Es besteht darüber hinaus die Möglichkeit, eigene Füllmuster zu entwerfen. Der Kreativität sind wieder einmal keine Grenzen gesetzt. Ein Muster ist 16 Bit breit, wir definieren es in einem UWORD-Array. Im einfachsten Fall bedeutet ein gesetztes Bit einen gesetzen Punkt und jedes Array-Element eine eigene Zeile im Muster. Die Höhe eines Musters entspricht also der Anzahl der Arrayelemente. Das Makro SetAfPt(RP, &data[0], n) teilt dem System unseren Wunsch mit. Die Parameter sind RastPort, ein Zeiger auf das »Muster-Array« und die Pixelhöhe des Musters.

Für die Höhe wird der Exponent zur Basis 2 (2^n) verlangt. Ist die gewünschte Höhe zum Beispiel 16, so müssen wir als Parameter für die Höhe den Wert 4 übergeben, denn 2^4 ergibt 16. Es kommen daher nur Höhen wie etwa 1, 2, 4, 8 etc. in Frage.

Die Farbe der gesetzten Punkte entspricht der des APen. SetAfPt() erlaubt aber auch die Definition mehrfarbiger Muster. Voraussetzungen: Sie müssen

- für jede Bitplane ein eigenes Muster entwerfen und
- den Exponenten, der die Höhe beschreibt, mit einem negativem Vorzeichen angeben.

Schauen wir uns noch einmal Listing 18 an. Einige Zeilen wurden mittels Kommentarzeichen von der Compilierung ausgeschlossen. Entfernen Sie jetzt bitte die Kommentarzeichen der beiden unteren Kommentare.

SetAfPt() erhält als aktuellen Parameter für die Höhe, die Hochzahl 2. Wir brauchen also ein 4 (2^2) Pixel hohes Muster für **jede** Bitplane (da der Exponent negativ ist). Der Workbenchscreen ist mit zwei Planes ausgestattet, demnach muß unser Array 2*4=8 Elemente enthalten. Die ersten vier stehen für das Muster der Plane 1, die nächsten vier für das der zweiten Plane. Wenn APen auf das oberste Farbregister eingestellt ist, auf dem Workbenchscreen ist dies Nummer 3, setzt SetAfPt() das Muster direkt in den Bitplanes. Entsprechend ist die Farbgebung (siehe Bild 1 im Kapitel »Screens«). Schauen Sie sich, nach erneuter Übersetzung, das Ergebnis an: Flood() füllt nun mit einem vierfarbigen Muster.

Eine weitere, überzeugend schnell arbeitende Einrichtung zur Erzeugung gefüllter Flächen stellen die sogenannten Area-Funktionen dar. Eine Anzahl gefüllter Kreise, Ellipsen und Polygone, die in der sogenannten Area-Liste aufgeführt sind, lassen sich mit einem einzigen Befehl auf den Bildschirm zeichnen.

Superschnell mit Area

Auch hier müssen wir ein wenig Vorarbeit leisten:
- Einrichten eines temporären Rasters und
- Initialisieren eines Area-Infos.

Den ersten Punkt haben wir gerade besprochen und den zweiten nimmt uns die Funktion InitArea() ab:

```
struct AreaInfo AInfo;
WORD Buffer[K*5];
InitArea(&AInfo,Buffer,
K);
RP->AreaInfo=&AInfo;
```

Die Parameter dieser Funktion von rechts nach links: das Symbol K stellt die Anzahl der Flächeneckpunkte (oder Mittelpunkt bei einem Kreis) plus 1 in einer Area-Liste dar. Demnach verlangt ein Dreieck die Angabe von vier und ein Kreis die Angabe von zwei Punkten. Beide zusammen in einer Area-Liste zählen 6 Punkte. Buffer ist die Adresse eines Speicherbereiches, der alle Koordinaten aufnehmen kann. Wie aus dem oben aufgeführten Programmfragment hervorgeht, errechnet sich der notwendige Speicherbedarf aus der Anzahl der Koordinaten mal 5.

InitArea() initialisiert die AreaInfo-Struktur an der Adres-

```
/*
\\      Brush.c - Zeichnen mit dem Teil eines RastPorts
*/
#include "TOOL.h"
#define RP_C (cli->RPort)
#define RP_W (win->RPort)
#define XPos (win->MouseX)
#define YPos (win->MouseY)

struct Window *win,*cli;

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung)  printf("Kann %s nicht öffnen!",meldung);
    if(win)      CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    ULONG idcmp=CLOSEWINDOW|MOUSEMOVE|MOUSEBUTTONS,
          flags=REPORTMOUSE|RMBTRAP|WINDOWCLOSE|ACTIVATE,
          class=0;
    USHORT code;
    UBYTE minterm;

    if(OpenLib()) error("Lib's");
    cli=ACTIVE_WINDOW;
    if(!(win=GetWindow(NULL,0,40,640,216,idcmp,flags,"Brush",NULL)))
              error("Window");
    while(class!=CLOSEWINDOW){          /* Programmende über
    CloseGadget */
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage(win,&code,0,0);
        if((class==MOUSEBUTTONS)&&((code==SELECTDOWN)||(code==
        MENUDOWN))){
            if(code==SELECTDOWN)     minterm=0xC0; /* Original
            übernehmen */
            else                     minterm=0x30;/* Original
            invertieren */
            while((code!=SELECTUP)&&(code!=MENUUP)){/* Knopf
            losgelassen */
                Wait(1<<win->UserPort->mp_SigBit);
                class=GetMessage(win,&code,0,0);
                if(class==MOUSEMOVE){
                    if(YPos>10)  /* Titelleiste mit CloseGadget
                    schützen */
                        ClipBlit(RP_C,0,0,RP_W,XPos,YPos,80,
                        40,minterm);
                    while((class=GetMessage(win,&code,0,0))==
                    MOUSEMOVE);
                }
            }
        }
    }
    error(0);
}
```

Listing 18. Der Blitter macht's möglich: Zeichnen mit dem Teil eines RastPorts als Brush.

```
struct BitMap
{
    UWORD BytesPerRow;         /* Bytes pro BitMap-Zeile (Breite/8) */
    UWORD Rows;                          /* Anzahl der Zeilen (Höhe) */
    UBYTE Flags;                                   /* ??? ungenutzt */
    UBYTE Depth;               /* Anzahl der BitPlanen (BitEbenen) */
    UWORD pad;     /* Platzhalter - PLANEPTR beginnen auf LONGWORD */
    PLANEPTR Planes[8]; /* die Adressen der BitEbenen im Speicher */
};
```

Listing 19. Elementare Daten über den Bildschirminhalt verwaltet BitMap

Verknüpfung	Minterm
!V & !N	00010000 = 0x10
!V & N	00100000 = 0x20
V & !N	01000000 = 0x40
V & N	00010000 = 0x80

Tabelle 8. Mit Minterms werden Daten verknüpft

se &AInfo. Wir weisen sie AreaInfo, einer weiteren Komponente der RastPort-Struktur, zu.

Die Vorbereitungen sind damit schon abgeschlossen. Es kann also losgehen: AreaMove (RP,x,y) schließt ein eventuell angefangenes Polygon und legt an der Position x/y des RastPorts RP den Anfangspunkt eines neuen fest.

AreaDraw(RP,x,y) erweitert das angefangene Polygon um den Eckpunkt x/y.

AreaEllipse(RP,x,y,rh,rv) hängt der Area-Liste Informationen über eine Ellipse an. Der Mittelpunkt liegt bei x/y, rh ist der horizontale Radius und rv der vertikale Radius. Das Makro AreaCircle(RP,x,y,r) zeichnet, wenn die Screenauflösung stimmt, eine Kreisfläche.

Area-Funktionen liefern, wenn alles klappt, den Wert 0, im anderen Fall 1. Ausnahme ist AreaDraw(), sie kehrt im Fehlerfalle mit -1 zurück. Das »Zauberwort«, das alle Flächen der Area-Liste buchstäblich auf einen Schlag darstellt, heißt AreaEnd(RP). Auch hier hilft SetAfPt(), wenn einfarbige Flächen zu langweilig wirken.

OPen zeichnet, nach einer entsprechenden Farbzuweisung, wie bei RectFill(), einen Rahmen um die Polygone. Dieses Extra läßt sich, wir haben es schon erwähnt, mit dem Makro BNDRYOFF(RP) abschalten.

Apropos RectFill(): Unser Area-Beispielprogramm befindet sich in Tool.h. Es handelt sich um eine Schwesterfunktion von RectFill().

EllFill(RP,col,x,y,rh,rv) zeichnet gefüllte Ellipsen. Parameter und Returnwert entsprechen bis auf »col«, der gewünschten Farbe, denen der Funktion AreaEllipse(). Das hiervon abgeleitete Makro CircFill() zeichnet ausgefüllte Kreise. Sie finden es in der Tool.h-Rubrik Makros.

Manchmal ist es notwendig, einen bestimmten Teil des RastPorts, vielleicht eine komplexe Zeichnung, an eine andere Stelle oder sogar in einen anderen RastPort zu kopieren. Auch hier stehen uns Routinen mit überraschender Ausführgeschwindigkeit zur Verfügung. Es handelt sich um die Blitter-Funktionen. Die Haupt- und Lieblingsaufgabe dieses Co-Prozessors ist das Kopieren von Daten, wobei er die Kopie gleichzeitig noch durch logische Verknüpfungen mit dem Zielbereich verändern kann.

Die im folgenden beschriebenen Funktionen nutzen diese Fähigkeit für den grafischen Bereich.

Beginnen wir mit ClipBlit (&von,x1,y1,&nach,x2,y2,b,h,m). Diese Funktion ermöglicht das Kopieren eines rechteckigen Bereiches, von RastPort »von« nach RastPort »nach«, in einer atemberaubenden Geschwindigkeit.

Die linke obere Ecke des Quellrechtecks befindet sich an der Position x1/y1 im RastPort »von«. Es ist »b« Pixel breit und »h« Pixel hoch. ClipBlit() kopiert es so, daß sich die linke Ecke an der Position x2/y2, im RastPort »nach« befindet. Natürlich können Sie für beide RastPorts auch die gleiche Adresse angeben.

Das Programm »Brush« (Listing 18) demonstriert die ClipBlit()-Funktion. Starten Sie das Programm aus dem CLI-Window. Sie können dann in einer eigenen Zeichenfläche mit einem Ausschnitt des CLI-Windows zeichnen. Drücken Sie die linke Maustaste, wird bei jeder Mausbewegung der Quellbereich unverändert in den Zielbereich kopiert. Mit Druck auf die rechte Taste erscheint das kopierte Rechteck invertiert. Wir haben die Möglichkeit der Veränderung des Bereichs durch eine logische Verknüpfung genutzt. Die Art der Verknüpfung teilen wir dem Blitter über den letzten Parameter »m« der ClipBlit()-Funktion mit. Die UBYTE-Variable »m« steht für Minterm. Tabelle 6 zeigt: mit den oberen 4 Bit teilen wir die gewünschte Verknüpfungsart mit.

Nehmen wir das einfachste Beispiel. Wert 0x80 als Minterm gibt als Ergebnis RastPort V (von) AND RastPort N (nach). Das heißt, nur wo in beiden RastPorts Punkte gesetzt sind, ist nach der Verknüpfung auch wirklich ein Punkt zu sehen. Ein weiteres Beispiel, diesmal mit einem Minterm-Wert von 0x40. Der Blitter setzt einen Punkt, wenn in V ebenfalls ein Punkt, aber in N kein Punkt gesetzt war. Die Verknüpfung erfolgt BitPlane für BitPlane. Die Farbe richtet sich nach dem Inhalt des Farbregisters mit der Nummer des neuen Wertes (siehe Bild 1). Sie können die in Tabelle 6

aufgeführten Minterms auch untereinander mit OR verknüpfen. Bei 4 Bits gibt es $2^4=16$ Möglichkeiten. Wieder einige Beispiele:

```
(!V&!N)|(!V&N)
```

Die Werte 0x10 und 0x20 wurden mit ODER zu 0x30 verknüpft. Der obige Ausdruck läßt sich noch vereinfachen, indem man !V ausklammert:

```
!V&(!N|N)=!V
```

Der Klammerausdruck !N:N wird zu 1 und wir erhalten demnach als Ergebnis !V&1, was !V entspricht.

Der Wert 0x30 invertiert also den Bereich in RastPort V und legt ihn in RastPort N ab. So wurde auch die Invertierung in Brush (Listing 19) erreicht.

Oder der Wert 0xC0:

```
(V&N)|(V&!N)=V
```

Hier wird der Quellbereich ohne Veränderung übernommen. Nehmen Sie sich doch das Programm Brush vor und testen Sie einmal alle 16 Minterms durch. Besonders interessant ist 0x60. Er veranlaßt den Blitter beide RastPorts mit EOR zu verknüpfen, das heißt nur an den Stellen werden Punkte gesetzt, wo entweder in V oder in N ein Punkt gesetzt war. Zweimaliges Blitten mit gleichen Parametern stellt den ursprünglichen Zustand wieder her. In Malprogrammen wird diese Möglichkeit oft genutzt, um selbstdefinierte Brushes an der Pointerspitze mitzuführen. ClipBlit() blittet von RastPort zu RastPort.

Gehen wir noch eine Ebene tiefer und arbeiten mit BitMaps. Oft ist es nicht notwendig, einen komplett initialisierten RastPort im Arbeitsspeicher mitzuschleppen. In manchen Fällen, zum Beispiel der UNDO-Buffer in Zeichenprogrammen, reichen Informationen über die Lage des Speicherplatzes für die einzelnen BitPlanes und deren Größe aus. Diese Informationen sind in der BitMap-Struktur (Listing 19) gebunden. Hier die Initialisierung einer BitMap-Struktur namens BM:

```
...
struct BitMap BM;
...
InitBitMap(&BM,p,b,h);
for(i=0;i<p;i++)>
BM.Planes[i]=
(PLANEPTR)AllocRaster
(b,h)));
if(BM.Planes[i])
BltClear
(BM.Planes[i],RASSIZE
(b,h),1);
else quit();
}
...
quit()
{
...
for(i=0;i<p;i++){
 if(BM.Planes[i])
FreeRaster(BM.Planes[i],
 b,h);
}
...
```

Mit der InitBitMap()-Funktion legen Sie fest, wie viele Planes (p) von der Größe b*h (Breite *Höhe) Pixel Sie benötigen. Die Funktion AllocRaster reserviert den notwendigen Speicherplatz im Chipmemory. Als Rückgabewert erhalten Sie die Adresse der Plane oder, falls ein Fehler auftrat, NULL. BltClear() löscht die angegebene Plane. Die notwendige Größe errechnen wir mit dem Makro RASSIZE().

Der dritte Parameter ist ein Flag mit folgender Bedeutung: ist Bit 1 gesetzt, deutet die Funktion die oberen 16 Bit des zweiten Parameters (hier RASSIZE(w,h) als die Anzahl der zu löschenden Zeilen und die unteren 16 Bit als die Anzahl der zu löschenden Bytes pro Zeile; ein Rechteck wird gelöscht. Ist jedoch Bit 1 gelöscht, so interpretiert BltClear() den zweiten Parameter als die tatsächliche Anzahl der zu löschenden Bytes. Setzt man Bit 0 auf eins, wartet das Programm, bis der Blitter mit Löschen fertig ist.

In unserem Beispiel gibt demnach der zweite Parameter die Anzahl der Bytes an, die in der BitPlane gelöscht werden sollen, dabei wartet das Programm so lange, bis der Blitter seine Arbeit verrichtet hat.

Es versteht sich von selbst, daß der disziplinierte Programmierer die reservierten BitPlanes vor Programmende wieder freigibt. Hier geschieht das mit FreeRaster(). Parameter sind neben der Adresse der Plane, deren Breite und Höhe. Das waren die Vorbereitungen, nun zur Anwendung:

```
struct BitMap von,nach;
SHORT x1,y1,x2,y2,b,h;
UBYTE m,M;
PLANEPTR t;
...
BltBitMap(&von,x1,y1,
&nach,x2,y2,b,h,m,M,t);
```

Diese riesige Anzahl von aktuellen Parametern sind notwendig, um ein Rechteck von einer BitMap in die andere zu »blitten«. Aber keine Angst, die Funktion ist der zuvor besprochenen sehr ähnlich. Die Parameter von und nach sind diesmal die Adressen der beiden BitMaps. Auch hier können beide gleich sein, nur müssen wir in diesem Fall Platz für einen Zwischenspeicher schaffen, falls sich die Bereiche überschneiden. Dieser Zwischenspeicher könnte zum Beispiel ein temporäres Raster, -wie bei den Area-Funkionen besprochen - sein. Es muß nur so groß sein, um den überlappenden Bereich aufnehmen zu können. Der letzte Parameter, im Beispiel t, verweist auf das Raster.

Das große M im Aufruf steht für Maske. Hiermit geben Sie an, welche BitPlane(s) die Funktion blitten soll. Für jede Plane müssen Sie ein Bit setzen, wobei die Nummer in der BitMap-Struktur gleich dem Bit in »Maske« ist.

Setzen Sie also Bit 0 und 1, wenn Sie möchten, daß die Planes 0 und 1 geblittet werden sollen. Ansonsten entsprechen die Parameter denen der bereits besprochenen ClipBlit()-Funktion.

Auch das Kopieren aus einer BitMap in einen RastPort ist möglich:

BltBitMapRastPort(&von,x1, y1,&nach,x2,y2,b,h,m). Die Adresse »von« ist die Quell-BitMap und »nach« die des Ziel-RastPorts. Alle anderen Parameter können Sie von den vorangegangenen Funktionen übernehmen.

Eine Funktion, die von einem RastPort in eine BitMap kopiert, existiert nicht. Vielleicht fällt Ihnen eine ein? Denken Sie dabei an die einzelnen Komponenten der RastPort-Struktur.

Graphics.library ist nicht nur für Grafik zuständig, sondern enthält noch eine Menge Funktionen, die die Textausgabe und das Arbeiten mit verschiedenen Zeichensätzen erleichtern oder überhaupt erst ermöglichen.

Der Kern der Print()-Funktion unseres Tools zum Beispiel, ist die Betriebssystemroutine Text(RP, &text, len) (siehe Listing 1). Diese gibt den Text an der aktuellen Grafikcursor-Position aus. Als Parameter dienen RastPort, ein Zeiger auf den Text und die Anzahl der Textzeichen.

Unsere Funktion Print(RP, &text, col, x, y) ermöglicht die einfache Ausgabe eines Textes an einer beliebigen Position des RastPorts. Dabei ist &text die Adresse, col die Farbe und x/y die Position des Ausgabetextes. Die Position bezieht sich, dank der Korrektur der Y-Position in der Print() Funktion, auf die linke obere Ecke des Textes. Normalerweise gibt Text() den Ausgabetext an der Baseline aus. Ein Aufruf wie

```
Print(RP,"Text",2, 1,
99);
Print(RP,"Text",1, 0,
100);
```

bewirkt eine Textausgabe mit »Schatten«. Die systemunabhängige Funktion strlen() ermittelt die Anzahl der Zeichen des Ausgabetextes. Dagegen liefert Ihnen TextLength(RP,&text,len) die Länge des Textes in Pixel als SHORT-Wert. Die Parameter entsprechen dabei denen der Text()-Funktion.

Jeden Zeichensatz können Sie im nachhinein softwaremä-

```
struct AvailFonts{
    UWORD af_Type; /* Speicher (AFF_MEMORY) oder Disk (AFF_DISK) */
    struct TextAttr af_Attr;            /* die TextAttr-Struktur */
};

struct TextAttr{
    STRPTR ta_Name;                     /* Name des Zeichensatzes */
    UWORD ta_YSize;                       /* die Höhe in Pixel */
    UBYTE ta_Style;                          /* die Schriftart */
    UBYTE ta_Flags;                /* verschiedene Einstellungen */
};
```

Listing 20. Wichtige Strukturen für die Verwendung neuer Zeichensätze

Stilvolle Schriften

Textstil	Ausgabe
FS_NORMAL	Normalschrift
FSF_UNDERLINED	Schrift unterstrichen
FSF_BOLD	Fettschrift
FSF_ITALIC	Kursivschrift
FSF_EXTENDED	Breitschrift

Tabelle 9. Die verschiedenen Schriftarten können kombiniert werden

ßig verändern. Die Text()-Funktion gibt dann die Zeichen nach Wunsch kursiv, unterstrichen, fett oder breit aus. Die einzelnen Schriftarten kann man natürlich auch kombinieren, zum Beispiel kursiv und unterstrichen.

Dies bewirkt ein Aufruf der Funktion SetSoftStyle(RP,nStil, mStil). Die Parameter sind RastPort, neuer Stil und möglicher Stil.

Tabelle 9 zeigt die möglichen Stil-Flags, die man als zweiter Parameter einträgt. Den dritten Parameter erhält man mit dem Aufruf

```
BYTE mStil;
mStil=AskSoftStyle(RP);
```

Schriften mit Stil

Er besagt, welche Schriftarten softwaremäßig erzeugt werden können. Der folgende Funktionsaufruf schaltet auf kursiv und unterstrichen:

```
SetSoftStyle(RP,FSF_
ITALIC|FSF_UNDERLINED,
AskSoftStyle(RP));
```

Natürlich kann man auch gänzlich neue Zeichensätze laden. Lassen Sie uns zuerst feststellen, welche Zeichensätze uns zur Verfügung stehen. Hierzu dient die Funktion:

```
fehler=AvailFonts
(&buffer,bytes,flags);
```

AvailFonts() erwartet als ersten Parameter die Adresse eines Buffers für die gewünschten Informationen. Die Größe dieses Buffers ist im Parameter bytes festgelegt. Mit Hilfe der Flags entscheiden wir, wo die Zeichensätze zu suchen sind. AFF_DISK sucht auf der Diskette im logischen Laufwerk »Fonts:« und AFF_MEMORY sucht im Speicher. Beide Flags kann man kombinieren.

Klappt die Funktion fehlerlos, so liefert sie den Wert 0. Im anderen Fall war der Buffer zu klein gewählt, der Returnwert gibt an, um wieviel Byte. Nach einer entsprechenden Berichtigung des Parameters bytes, sollte dann alles funktionieren.

Nach erfolgreichem Aufruf finden wir an der Adresse &buffer die Struktur AvailFontHeader. Sie enthält nur eine einzige Komponente:

```
WORD afh_NumEntries
```

die Anzahl der gefundenen Zeichensätze. Gleich hinter dieser Struktur im Buffer ist für jeden Zeichensatz eine AvailFonts-Struktur (Listing 20) angelegt. Diese enthält neben dem »Fundort« (AFF_DISK, AFF_MEMORY), die TextAttr-Struktur. Wie die TextAttr-Struktur aufgebaut ist und welche Bedeutung die einzelnen Komponenten haben, geht ebenfalls aus Listing 20 hervor. SetFont(RP,&TextFont) aktiviert einen neuen Zeichensatz. Neben der Angabe des RastPorts verlangt diese Funktion die Adresse einer TextFont-Struktur, wie sie im Headerfile graphics/text.h definiert ist.

Für Zeichensätze, die im Speicher abgelegt sind, erhalten wir solch einen Zeiger mit dem Aufruf der Funktion OpenFont(&TextAttr). Für Disk-Fonts ist OpenDiskFont(&TextAttr) zuständig, allerdings muß hierfür die Diskfont-Library geöffnet sein:

```
ULONG *DiskfontBase;
DiskfontBase=(ULONG *)
OpenLibrary("
diskfont.library",0);
```

Beide Male übergeben wir einen Zeiger auf die TextAttr-Struktur und erhalten einen Zeiger auf die TextFont-Struktur zurück.

Dabei spielt es keine Rolle, ob Sie die Adresse einer selbst vereinbarten TextAttr-Struktur oder die einer mittels AvailFonts() erzeugten übergeben.

Die Funktion AskFont(RP,&TextAttr) initialisiert die TextAttr-Struktur an der angegebenen Adresse mit den Informationen des aktuellen Zeichensatzes. Ein Aufruf der Funktion AddFont(&TextFont) hängt der Zeichensatz-Liste einen neuen Zeichensatz an. Das Gegenstück dazu, RemFont(&TextFont), löscht den entsprechenden Zeichensatz aus der Liste.

Ein nicht mehr benötigter Zeichensatz belegt so lange wertvollen Speicherplatz, bis man ihn mit CloseFont(&TextFont) endgültig verabschiedet.

Pointer für jedes Window

Die Funktion SetPointer(&Window,&data[0],h,b,x,y) erlaubt die Definition eines eigenen Mauszeigers im angegebenen Window. Dieser Zeiger ist immer dann sichtbar, wenn sich das Window aktiviert (siehe auch Listing 21).

Die Parameter dieser Funktion von links nach rechts: die Adresse des Fensters, in dem der neue Pointer sichtbar sein soll, die Adresse des Datenblocks, der die Pointerdaten enthält, die Höhe des Pointers, die auf 16 Pixel beschränkte Breite und die x/y-Koordinate der linken oberen Ecke des so-

```
/*
\\    Snake - demonstriert Pointer & Sprites
*/
#include "TOOL.h"
#define VP ACTIVE_SCREEN->ViewPort
#define RP win->RPort

SHORT xpos,ypos,lastx[90],lasty[90],tail,len,points;
UWORD *CP, *CB;
UWORD Pointer[]={
    0,0,   0xFFFF,0, 0,0xFFFF, 0xFFFF,0xFFFF,   0,0
};
UWORD Ball[]={
    0,0,   0x1800, 0x2000, 0x1C00,0x6000, 0x8C00,0xF200,
    0x8000,0xFE00, 0xC200,0xFE00, 0x7C00,0x7C00,
    0x3800,0x3800, 0,0
};
struct Window *win;
struct SimpleSprite Sprite={
    NULL,7,10,10,2
};

VOID error(meldung)
    STRPTR meldung;
{
    if(meldung) printf("Kann %s nicht öffnen",meldung);
    if(CP)      FreeMem(CP,sizeof(Pointer));
    if(CB)      FreeMem(CB,sizeof(Ball));
    if(win)     CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID Score()
{/* Spielstandsanzeige in Form eines Balkens */
    SHORT col=points>8?3:2,
          top=win->Height-((win->Height-10)/10)*points;
    Block(RP,1,0,10,10,win->Height);
    Block(RP,col,2,top,8,win->Height-2);
}

VOID Init()
{/* ein neues Spiel beginnt */
    SHORT i;

    tail=points=0;
    xpos=ypos=10;
    len=(win->Width+win->Height)/10;        /* Länge von
    Windowgröße abhängig */
    for(i=0;i<len;i++) lastx[i]=lasty[i]=0;
    SetRast(RP,0);                                  /* RastPort
    löschen */
    RefreshWindowFrame(win);        /* Rahmen, Titelleiste, Gadgets
    refreshen */
    Score();
}

VOID main()
{
    SHORT  i,x=2,y=1,SpriteNr,tip,col=2;
    USHORT code;
    ULONG  class=0,idcmp=CLOSEWINDOW|NEWSIZE|MOUSEBUTTONS,
           flags=WINDOWCLOSE|WINDOWDEPTH|WINDOWSIZING|RMBTRAP
           |ACTIVATE;

    if(OpenLib()) error("Lib's");
    if(!(win=GetWindow(NULL,0,0,640,80,idcmp,flags,"Snake",0))) error
    ("Window");
    if((CP=AllocMem(sizeof(Pointer),MEMF_CHIP))==0) error("Chip-
    Memory");
    if((CB=AllocMem(sizeof(Ball),MEMF_CHIP))==0)    error("Chip-
    Memory");
    for(i=0;i<10;i++) CP[i]=Pointer[i];     /* Daten ins Chip-Memory
    kopieren */
    for(i=0;i<18;i++) CB[i]=Ball[i];
    Sprite.posctldata=&CB[0];               /* Adresse Sprite-Daten
    bekanntgeben */
    SetRGB4(&VP, 21,12,12,12);
    SetRGB4(&VP, 22,10,10,10);
    SetRGB4(&VP, 23, 8, 8, 8);
```

Listing 21. Spiele sind das Hauptanwendungsgebiet für Sprites

```
        SetPointer(win,&CP[0],3,16,0,0);          /* Mauspointer verändern */
        SpriteNr=GetSprite(&Sprite,2);
        Init();
        while(class!=CLOSEWINDOW){
            class=GetMessage(win,&code,0,0);
            if(class==NEWSIZE) Init();          /* Fenstergröße wurde
            verändert */
            MoveSprite(&VP,&Sprite,xpos+=x,ypos+=y);     /* neue Sprite-
            Position */
            lastx[tail]=xpos+8-x;                   /* 1. Position des
            Schweifs */
            lasty[tail]=ypos+2-y;
            SetDrMd(RP,JAM1|COMPLEMENT);
            DrawEllipse(RP,lastx[tail],lasty[tail],2,1);    /* Kopf des
            Schweifs */
            tail=tail>len-2?0:tail+1;
            tip =tail>len-2?0:tail+1;               /* Ende des Schweifs
            löschen: */
            if(lastx[tip]+lasty[tip]) DrawEllipse(RP,lastx[tip],
            lasty[tip],2,1);
            if(xpos>=win->Width-16) x=-2;               /* Bewegung
            nach links */
            else if(xpos<=10)      x= 2;              /* Bewegung
            nach rechts */
            if(ypos>=win->Height-8){
                y=-1;                             /* Bewegung nach oben */
                points++;
                if(points>9){
                    SetWindowTitles(win,">> GAME-OVER <<
                    Klick...",-1);
                    do{
                        Wait(1<<win->UserPort->mp_SigBit);
                        class=GetMessage(win,&code,0,0);
                    }while((code!=SELECTUP)&&(class!=CLOSEWINDOW));
                    SetWindowTitles(win,"Snake",-1);
                    Init();
                }
                else Score();
            }
            else if(ypos<=10) y=1;                      /* Bewegung
            nach unten */
            if((win->MouseY>=ypos+9)&&(win->MouseY<=ypos+11))
            {       /* Schläger */
                if((y>0)&&(xpos>win->MouseX-12)&&(xpos<
                win->MouseX+28)) y=-1;
            }
        }
        FreeSprite(SpriteNr);
        error(0);
}
```

Listing 21. (Schluß)

genannten Hot-Spots. So bezeichnet man die empfindliche Stelle im Mauszeiger, mit der Sie zum Beispiel ein Workbenchobjekt anwählen können.

Unklar ist noch das Aussehen des Datenblocks. Es handelt sich um ein UWORD-Array, dessen beiden ersten und beiden letzten Elemente gleich 0 sein müssen.

Zwischen diesen, für das System reservierten Arrayelementen, befindet sich die »Beschreibung« des Pointersprites. Sie haben richtig gelesen, der Mauszeiger ist nichts anderes als einer der acht zur Verfügung stehenden Hardwaresprites. Er trägt die Nummer 0. Ein Hardwaresprite besteht aus maximal drei Farben. Aus Tabelle 3 geht hervor, daß immer zwei Sprites auf die gleichen Farbregister zugreifen. Die Farben des Pointers sind in den Farbregister 17-19 zu finden. Das oben angesprochene Datenarray enthält als Beschreibung für jede Zeile des Pointers zwei hintereinanderliegende, 16 Bit breite Werte.

Stellen Sie sich nun bitte die beiden Werte in binärer Form untereinander vor, etwa so:

```
%1100101001110011
%1001111001100110
```

Lesen Sie die senkrecht übereinanderliegenden Bits von unten nach oben. Liegen zwei Nullen (00) übereinander, so ist der Sprite an dieser Stelle der Zeile durchsichtig, der Punkt ist nicht gefärbt. Finden Sie (von unten nach oben) die Bits 01, so ist dieser Punkt in der Farbe des ersten (%01=1) zuständigen Registers gefärbt (Tabelle 10). Die Bits 10 verweisen auf das zweite (%10=2) und schließlich die Kombination 11 auf das dritte (%11=3) Farbregister.

Schauen wir uns die Daten des Pointers in Listing 21 an. Sie finden sie gleich am Anfang des Listings im Array Pointer[].

Die erste Zeile, bestehend aus 0xFFFF und 0, ist in der Farbe des Registers 17 gefärbt (%01), die zweite Zeile, 0 und 0xFFFF, in der des Registers 18 (%10) und die dritte Zeile, zweimal 0xFFFF, in der des Farbregisters Nummer 19 (%11).

Der drei Zeilen hohe Pointer enthält keine transparente Zeile (%00). Vor und hinter den notwendigen sechs Elementen sind jeweils die beiden 0-UWORDs angegeben.

Beachten Sie, daß alle Spritedaten, also auch die des Pointers, im CHIP-Memory liegen müssen.

Bleiben wir doch noch bei den Sprites. Diese Minigrafiken, Sie kennen Sie aus Spielen, können völlig frei auf dem Bildschirm bewegt werden. Listing 21 verdeutlicht das Arbeiten mit SetPointer() und Sprites. »Snake« ist ein kleines Spiel. Ein Sprite, in Form eines Balls, reflektiert an den Seiten des Windows, dabei zieht er einen Schweif hinter sich her. Ihre Aufgabe ist es, dafür zu sorgen, daß der Ball den Window-Boden nicht berührt.

Als Hilfsmittel steht Ihnen dafür der als Break-Out-Schläger modifizierte Pointer zur Verfügung. Springt der Ball doch einmal auf den Boden, so wächst die Game-Over-Säule am linken Feldrand ein wenig. Diese Säule färbt sich rot und warnt Sie, wenn das Spielende kurz bevorsteht. Die Länge des Schweifs ist abhängig von der Fenstergröße.

Wie weiter oben bereits angedeutet, können wir auf dem Amiga acht Hardware-Sprites definieren. Wenn Sie alle Sprites so auf den Bildschirm bringen, daß sie sich teilweise verdecken, ist das Sprite mit der niedrigeren Nummer immer vor dem mit der höheren Nummer zu sehen. Diese Prioritäten sind festgelegt und nicht veränderbar.

Sprites dürfen maximal 16 Pixel breit, aber beliebig hoch sein. Die Größe der Pixel ist von der Screenauflösung unabhängig. Ein Spritepixel entspricht immer einem Bildschirm-Pixel im LoRes-Modus. Ein Screen muß für die Verwendung von Sprites vorbereitet sein, das geschieht durch das Setzen des SPRITES-Flags in der NewScreen-Strukturkomponente ViewModes. Wie aus Listing 21 hervorgeht, ist der Workbenchscreen schon vorbereitet.

Nach der Vereinbarung verschiedener Variablen, erfolgt in Snake.c die Initialisierung der SimpleSprite-Struktur. Sie besteht aus folgenden Komponenten: einem Zeiger auf die im CHIP-Memory liegenden Spritedaten, die Höhe des Sprites, die x/y-Position und die Nummer des Sprites.

Das Aussehen der Spritedaten haben wir gerade bei der Veränderung des Mauszeigers besprochen. Beachten Sie bitte, daß Sie für jedes Sprite einen eigenen Datenblock vereinbaren müssen, auch wenn Sie mit mehreren gleich aussehenden Sprites arbeiten möchten. Die Struktur SimpleSprite ist im Includefile graphics/sprite.h definiert. Bevor wir unser Sprite auf dem Bildschirm darstellen können, müssen wir es noch »reservieren«:

```
SpriteNr=GetSprite
(&Sprite,2);
```

Nicht nur für Spieler: Sprites

GetSprite() verlangt als Parameter einen Zeiger auf die SimpleSprite-Struktur und die gewünschte Spritenummer. Ist das Sprite mit dieser Nummer noch frei, so entspricht auch der Returnwert dieser Nummer. GetSprite() liefert -1, wenn das Sprite bereits reserviert war.

Bei der Angabe von -1 als Spritenummer liefert die Funktion die Nummer des nächsten freien Sprites. Für die Darstellung und die Bewegung der Sprites ist MoveSprite(&VP, &SSprite,x,y) zuständig. VP ist dabei ein Zeiger auf die ViewPort-Struktur des Screens, SSprite zeigt auf die SimpleSprite-Struktur und x/y ist die gewünschte Position des Sprites.

ChangeSprite(&VP,&SSprite, &data[0]) verändert das Aussehen des Sprites mit der angegebenen SimpleSprite-Struktur. Der dritte Parameter zeigt auf das Array mit den neuen Spritedaten.

Ein einzelnes Sprite läßt sich mit FreeSprite(SpriteNr) löschen. Der Parameter ist die von GetSprite() gelieferte Spritenummer. Das Makro OFF_SPRITE läßt alle Sprites verschwinden, indem es einfach den DMA-Kanal abstellt. Das Gegenstück ON_SPRITE stellt den Urzustand wieder her.

Wie aus Tabelle 10 ersichtlich ist, greifen immer zwei Sprites auf die gleichen Farbregister zu. Ein solches Sprite-Paar kann man zu einem einzigen Sprite verbinden, einem sogenannten Attached-Sprite. Beide Sprites müssen wir an die gleiche Bildschirmposition bringen. Da eine Zeile eines Sprites durch zwei UWORDs darge-

Hardwaresprites

Sprites	Register für Farbe 1	2	3
0 und 1	17	18	19
2 und 3	21	22	23
4 und 5	25	26	27
6 und 7	29	30	31

Tabelle 10. Die Farbregisternummern der Sprites

Ansteuern der BitPlanes

Wert dez	Wert bin	BitPlane
0	00	keine
1	01	Nummer 0
2	10	Nummer 1
3	11	Nummer 0 und 1

Tabelle 11. Die Auswahl der Bitplanes erfolgt mittels PlanePick und PlaneOnOff

```
struct Border
{
    SHORT LeftEdge, TopEdge; /* Position der linken oberen Ecke des
    Rahmens */
    UBYTE FrontPen, BackPen,                     /* APen und BPen */
          DrawMode;                            /* der Zeichenmodus */
    BYTE Count;                        /* die Anzahl der Eckpunkte */
    SHORT *XY;     /* Adresse des Arrays, mit den Koordinaten der
    Eckpunkte */
    struct Border *NextBorder;          /* Adresse der nächsten
    Borderstruktur */
};
```

Listing 22. Die Borderstruktur ist in »intuition.h« definiert

```
struct Image
{
    SHORT LeftEdge, TopEdge,   /* Position der linken oberen Ecke des
    Images */
          Width, Height, Depth;    /* ... und dessen Breite, Höhe
          und Tiefe */
    USHORT *ImageData;                          /* die Adresse der
    Imagedaten */
    UBYTE PlanePick, PlaneOnOff;              /* Verknüpfung mit den
    Bitebenen */
    struct Image *NextImage;              /* die Adresse eines weiteren
    Images */
};
```

Listing 23. Die Image-Struktur enthält Informationen über das Aussehen einer Kleingrafik

stellt wird, liegen bei zwei Sprites vier UWORDs übereinander. Mit 4 Bit, die einem Pixel eines Attached-Sprites entsprechen, können wir 16 Zustände darstellen. Das ergibt unsere 15 Farben, gespeichert in den Farbregistern 17-31 und transparent, wenn alle Bits auf Null gesetzt sind.

Die beiden höherwertigen Bits stammen von dem Sprite mit der höheren Nummer. Die Farben werden sichtbar, wenn Sie in beiden Spritedaten-Arrays das Attached-Bit setzen: Data[1]=SPRITE_ATTACHED.

Intuition und Grafik

Auch Intuition verfügt über einige interessante Grafikfunktionen. In diesem Abschnitt besprechen wir die Borders, die Images, Intuition-Text und die Veränderung des Mauszeigers.

Die Handhabung dieser Funktionen scheint vielleicht zu Beginn etwas umständlich, es müssen beispielsweise vor dem ersten Aufruf Strukturen und Arrays definiert werden. Bei näherem Hinsehen ist aber – wie immer – alles »halb so wild«.

Beginnen wir mit den Borders. Der Name läßt das Einsatzgebiet erahnen: das Zeichnen von Umrahmungen. Die Funktion »DrawBorder(RP, &border,x,y)« zeichnet einen oder mehrere Rahmen. Parameter sind, neben der Adresse des RastPorts RP und der x/y-Position der linken oberen Ecke des Rahmens, ein Zeiger auf die Borderstruktur. Listing 22 zeigt diese Struktur mit der Bedeutung der einzelnen Strukturkomponenten. Left- und TopEdge, Front- und BackPen sowie der Zeichenmodus sind Ihnen mittlerweile ein Begriff. Count ist die Anzahl der Eckpunkte, einschließlich Anfangs- und Endpunkt. Ein geschlossenes Viereck hat demnach den Count-Wert 5.

Die einzelnen x- und y-Werte der Koordinaten sind in einem SHORT-Array gebunden. Ein Beispiel für unser Viereck:

```
SHORT XY[]={
  100,100,
  150,100,
  150,150,
  100,150,
  100,100
};
```

Der erste Wert ist die x-Koordinate des Anfangspunktes, der zweite Wert dessen y-Koordinate etc.

Die Werte des Anfangspunktes addieren sich zu den Parametern x beziehungsweise y der DrawBorder()-Funktion und zu den Variablen Left- beziehungsweise TopEdge der Borderstruktur.

Die Adresse des Koordinaten-Arrays tragen wir in das nächste Element der Border-Struktur ein: SHORT *XY. Möchten Sie gerne mehrere Rahmen miteinander kombinieren, verketten Sie die Strukturen, indem Sie die letzte Komponente mit einem Zeiger auf die folgende Border-Struktur belegen. Ansonsten tragen Sie dort NULL ein. Wir begegnen der Borderstruktur noch einmal weiter unten, wenn wir uns mit den Gadgets beschäftigen.

Auch die Images werden dort noch einmal interessant. Images sind kleine Grafiken, die beispielsweise bei einem Gadget auf dessen Funktion hindeuten, zum Beispiel die Depth-Gadgets.

Die Definition eines Images erinnert an die eines Borders: Wir müssen eine Image-Struktur initialisieren, ein USHORT-Array enthält die Daten des Images und die Funktion DrawImage(RP,&image,x,y) schließlich, bringt die Kleingrafik auf den Bildschirm.

Schauen wir uns zuerst die Image-Struktur (Listing 23) an. Nach den »alten Bekannten«, wie Position und Dimension, müssen wir mit der Variablen Depth die Tiefe, das heißt die Anzahl der Image-Bit-Ebenen, festlegen. Natürlich kann Depth die in der NewScreen-Struktur vereinbarte Anzahl der BitEbenen nicht überschreiten.

Es folgt die Adresse der Image-Daten. Für jede der unter Depth vereinbarten BitPlanes muß eine eigene Bildbeschreibung in Form eines Datenpakets in einem USHORT-Array zur Verfügung gestellt werden.

Betrachten wir zunächst Images mit einer einzigen BitPlane, die eine Breite von 16 Pixel nicht überschreiten.

Die Pixelhöhe, und somit auch die Variable Height der Image-Struktur, einer solchen Kleingrafik ist **in diesem Fall** identisch mit der Anzahl der Elemente des Arrays. Jedes Arrayelement beschreibt eine Zeile der Grafik. Ein gesetztes Bit in einem Element bedeutet ein gesetzter Punkt in einer Zeile.

Das Array enthält sozusagen das Bitmuster unserer Grafik. Die Variable Width enthält die Image-Breite in Pixel. Setzen wir Depth auf 2, arbeiten wir also mit zwei BitPlanes, müssen wir die Anzahl der Array-Elemente verdoppeln. Die zweite Hälfte der Elemente beschreibt dabei das Bitmuster der zweiten BitPlane. Bildlich gesprochen liegt diese über der ersten BitPlane.

Wir reduzieren nun Depth wieder auf eins und benutzen eine einzige BitPlane. Dafür soll unsere Grafik 21 Pixel breit sein. Wir müssen die Variable Width entsprechend einstellen.

Beachten Sie, daß Height nur eine Aussage über die Höhe der Grafik macht, nicht aber über die Anzahl der Array-Elemente. Jetzt sind **zwei** Elemente pro Grafik-Zeile nötig. Das erste Element beschreibt die ersten 16 Bit der Zeile und die 5 höchsten Bit des zweiten Elements den Rest.

Mit Hilfe der Variablen PlanePick und PlaneOnOff können wir entscheiden, mit welchen BitPlanes wir arbeiten möchten.

PlanePick bestimmt die Plane, in der die Grafik erscheinen soll. Dagegen wird jedes Pixel der mit PlaneOnOff gewählten Plane auf 1 gesetzt. Die BitPlanes selektiert man durch Setzen einzelner Bits in PlanePick beziehungsweise PlaneOnOff.

Tabelle 11 zeigt die möglichen Werte für das Arbeiten mit zwei BitPlanes. Die passenden

```
/*
\\    Smily - demonstriert Border und Image
*/
#include "TOOL.h"
#define RP   win->RPort
#define XPOS win->MouseX
#define YPOS win->MouseY

USHORT *CData;
USHORT FData[]={                                    /* Image-Daten */
    0,0,0,0,252,0,771,0,1024,32768,2048,16384,4096,8192,
    4300,8192,4402,8192,8192,4096,8192,4096,9729,36864,
    8705,4096,4354,8192,4348,8192,4216,8192,2048,16384,
    1024,32768,771,0,252,0,0,0,0,0
};
SHORT RData[2][10]={                                * Border-Daten */
    0,0, 40,0, 40,40, 0,40, 0,0,
    0,0, 30,0, 30,50, 0,50, 0,0
};
struct Image Bild      ={
    9,15, 22,22,1, NULL, 2,1, NULL
};
struct Border Rahmen[2]={
    0,5, 1,0, JAM1, 5, &RData[0][0], NULL,
    5,0, 3,0, JAM1, 5, &RData[1][0], &Rahmen[0]
};

VOID error(Meldung)
    STRPTR Meldung;
{
    if(Meldung)  printf("Kann %s nicht öffnen",Meldung);
    if(CData)    FreeMem(CData,sizeof(FData));
    CloseLib();
    exit(0);
}

VOID main()
{
    struct Window *win;
    SHORT i,x,y;
    USHORT code=0;

    if(OpenLib()) error ("Lib's");
    if((CData=AllocMem(sizeof(FData),MEMF_CHIP))==0) error("Chip-
    Memory");
    for(i=0;i<44;i++) CData[i]=FData[i];        /* Image-Daten ins
    CHIP-Memory */
    Bild.ImageData=CData;              /* neue Adresse der Daten im
    CHIP-Memory */
    win=ACTIVE_WINDOW;            /* Arbeiten im gerade aktiven
    (CLI) Fenster */
    ModifyIDCMP(win,MOUSEBUTTONS|MOUSEMOVE);/* CLI-Fenster reagiert
    auf Maus */
    while(code!=SELECTDOWN){                          /* Warten auf
    Mausklick */
        GetMessage(win,&code,0,0);
        DrawImage(RP,&Bild,XPOS,YPOS);                 /*
        Minigrafik zeichnen */
        DrawBorder(RP,&Rahmen[1],XPOS,YPOS);    /* doppelter
        Rahmen zeichnen */
    }
    ModifyIDCMP(win,0);
    error(0);
}
```

Listing 24. »Smily.c« arbeitet mit Border und Image

Werte für einen Screen mit mehr als zwei BitPlanes lassen sich daraus ableiten.

Listing 24 sollte die letzten Fragen beseitigen. Starten Sie »Smily« aus dem CLI-Fenster. An der aktuellen Mausposition erscheint ein mit Border eingerahmter »Smily«. Solange Sie die linke Maustaste nicht betätigen, erscheint, immer wenn Sie die Maus bewegen, die Figur an der Spitze des Pointers. Können Sie durch einen einzigen Funktionsaufruf erreichen, daß das Programm vor jedem Neuzeichnen die alte Figur löscht? Verändern Sie ruhig einmal die einzelnen Variablen der Strukturen und Arrays in »Smily.c«, um zu sehen, wie sich daraufhin die Grafik verändert.

Zwei Funktionen in diesem Programm haben wir noch nicht besprochen:

```
Speicher=AllocMem(Bytes,
Flags);
FreeMem(Speicher,Bytes);
```

AllocMem() reserviert einen Speicherbereich von der Größe; die der erste Parameter be-

```
/*
\\    AMem - Kontrolle des Arbeitsspeichers
*/
#include "TOOL.h"

struct Window *win;
UBYTE mem[40];
struct IntuiText IText={
    1,0,JAM2,0,0,NULL,(UBYTE *)mem,NULL
};

VOID error(Meldung)
    STRPTR Meldung;
{
    if(Meldung) printf("Kann %s nicht öffnen",Meldung);
    if(win)     CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    ULONG flags=WINDOWDEPTH|WINDOWDRAG|WINDOWCLOSE,class;

    if(OpenLib()) error ("Lib's");
    if(!(win=GetWindow(NULL,273,0,317,10,CLOSEWINDOW,flags,
    NULL,NULL)))
        error("Window");
    do{
        sprintf(mem," Chip: %d  Fast: %d ",AvailMem(MEMF_CHIP),
        AvailMem(MEMF_FAST));
        PrintIText(win->RPort,&IText,30,1) ;
        Delay(30);
        class=GetMessage(win,0,0,0);
    }while(class!=CLOSEWINDOW);
    error(0);
}
```

Listing 25. Prüfen Sie mit AMem, ob Ihr Programm alle Ressourcen freigibt. Starten Sie das Programm bitte mit »RUN AMem«.

schreibt. Die Flags sind im Headerfile »exec/memory« vereinbart, weshalb wir dieses File in Tool.h unter Headerfiles eingefügt haben.

Folgende Flags sind möglich: »MEMF_PUBLIC« bedeutet, daß der Speicherbereich nach der Zuweisung nicht mehr verschoben werden darf. »MEMF_CLEAR« füllt den Speicherbereich vor der Zuweisung mit Nullen.» MEMF_FAST« legt den Speicherbereich außerhalb des Chip-Memory. Dies ist natürlich nur mit einer RAM-Erweiterung sinnvoll. »MEMF_CHIP« legt den Speicherbereich ins Chip-Memory.

Die sogenannten Custom-Chips, beispielsweise der Blitter, können nur auf Daten zugreifen, die sich im Chip-Memory befinden. Dies sind zur Zeit die untersten 512 KByte des gesamten Speichers.

Bei der Verwendung einer RAM-Erweiterung werden möglichst viele Daten im Fast-Memory gespeichert, um das Chip-Memory zu entlasten. Wir müssen also, um sicher zu gehen, unsere Image-Daten, auf die der Blitter Zugriff haben muß, in diesen Speicherbereich kopieren (siehe Listing 26).

FreeMem() gibt später den Speicherbereich wieder frei. Parameter sind dabei die Adresse des Speicherblocks und deren Größe in Bytes.

Wie bereits mehrfach angesprochen, ist es sehr wichtig, beim Programmende den benutzten Speicher wieder freizugeben. Um eine bessere Kontrolle zu haben, vergleichen Sie den freien Speicher vor dem Aufruf und nach Beendigung Ihres Programms. Eine genaue Speicheranzeige liefert AMem (Listing 25). Binden Sie es am besten in die Startup-Sequence ein:

```
RUN AMem
```

AMem gibt die Größe noch verfügbaren Speichers, unterteilt in Chip- und Fastmemory, in einem eigenen Window aus.

Weiter oben haben wir die Textfunktionen aus der Graphics-Library besprochen. Unter Intuition finden wir ebenfalls eine Textfunktion.

PrintIText(RP,&intuitext,x,y) schreibt einen Text an die angegebene Position des RastPorts. In Listing 25 geben wir so die ermittelten Werte für Chip- und Fastmemory aus. Der zweite Parameter verweist jedoch

```
struct IntuiText
{
    UBYTE FrontPen, BackPen, DrawMode; /* Farben und
    Zeichenmodus */
    SHORT LeftEdge, TopEdge;            /* die Position des Textes */
    struct TextAttr *ITextFont;                   /* der
    Zeichensatz */
    UBYTE *IText;                          /* der auszugebende Text */
    struct IntuiText *NextText; /* die nächste IntuiText-Struktur */
};
```

Listing 26. Textausgabe unter Intuition: die IntuiText-Struktur.

```
struct IODRPReq {
    struct Message io_Message;
    struct Device *io_Device;
    struct Unit *io_Unit;
    UWORD io_Command;              /* das Kommando; hier DumpRastPort */
    UBYTE io_Flags;
    BYTE io_Error;                           /* IO-Error (siehe Tabelle 10) */
    struct RastPort *io_RastPort;           /* Adresse des RastPorts */
    struct ColorMap *io_ColorMap;            /* Adresse der ColorMap */
    ULONG io_Modes;           /* der Screenmodus, zum Beispiel HIRES */
    UWORD io_SrcX;       /* linke Seite des auszugebenden Rechtecks */
    UWORD io_SrcY;/* obere Seite des Rechtecks, bezogen auf Screen */
    UWORD io_SrcWidth;        /* die Breite des Bereichs im RastPort */
    UWORD io_SrcHeight;         /* die Höhe des Bereichs im RastPort */
    LONG io_DestCols;    /* die Breite des Bereichs auf dem Drucker */
    LONG io_DestRows;      /* die Höhe des Bereichs auf dem Drucker */
    UWORD io_Special;     /* die speziellen Druck-Modi (Tabelle 11) */
};
```

Listing 27. In der IODRPReq-Struktur muß man den RastPort beschreiben, den man ausdrucken möchte

Die Druckerfehler

FehlerNr	Bedeutung
1	Abbruch durch User
2	kein Drucker angeschlossen
3	keine Invertierung im HAM-Modus
4	fehlerhafte Koordinaten
5	Ausdruck in dieser Größe nicht möglich
6	kein Speicher für Berechnungen
7	kein Speicher für Druckerpuffer

Tabelle 12. Die Drucker-Fehlermeldungen io__Error der IODRPReq-Struktur

nicht auf einen Ausgabetext, sondern auf eine IntuiText-Struktur (Listing 26).

Diese Art der Textausgabe ist sehr komfortabel, doch gewinnt sie erst volle Bedeutung, wenn Sie etwas mehr über Gadgets und Menüs wissen.

Vermutlich haben Sie mit den oben beschriebenen Funktionen genug zu tun. Wir möchten dennoch nicht versäumen, Ihnen ein paar Aufgaben zu stellen:

- Zeichnen Sie Punkte auf einem RastPort. Die X-Position soll vom Mauszeiger, die Y-Position vom Elektronenstrahl der Bildröhre abhängig sein.
- Verändern Sie »Flow.c« so, daß der Text mit Schatten ausgegeben wird. Die SoftStyle-Schriftart soll frei wählbar sein (Kommandoparameter). Bei Betätigung der rechten Maustaste soll der Text mit halber Geschwindigkeit scrollen. Verändern Sie das Programm so, daß es mit verschiedenen (wählbaren) Zeichensätzen läuft.
- Erweitern Sie das Spiel Snake um zusätzliche Bälle. Bauen Sie Hindernisse auf, an denen die Bälle reflektieren oder die bei einer Kollision verschwinden (Tip: zur Kollisionsüberwachung können Sie ReadPixel() einsetzen).
- Schreiben Sie ein kleines Malprogramm mit verschiedenen Farben und den bekannten Funktionen wie Draw, Line, Circle, Block, Fill, Text etc.

Vielleicht möchten Sie gerne Ihre Zeichnungen zu Papier bringen oder benötigen Sie für Ihr Malprogramm noch eine Hardcopy-Routine? Lassen Sie uns an dieser Stelle, bevor wir dann endgültig zu Intuition zurückkehren, die Tool.h-Funktion HrdCpy() besprechen.

Tool.h macht Druck

Der Weg zur Hardcopy führt über das Printer-Device. Um diesen Weg zu verstehen, müssen Sie natürlich zuerst wissen, was ein Device ist.

Device heißt - wörtlich übersetzt - Gerät. Es handelt sich hier aber nicht um Geräte, wie zum Beispiel das Diskettenlaufwerk oder den Drucker, sondern eher um Routinensammlungen ähnlich einer Library. Über Device können wir mit den gewünschten Geräten in Verbindung treten. Voraussetzung dafür ist, daß wir das Device korrekt öffnen. Dies geschieht, ähnlich wie bei den Libraries, mit einer Open-Funktion:

```
Error=OpenDevice(&name,
u,&io,flag)
```

Die Variable name steht für den Device-Namen) in unserem Fall also "printer.device" (Listing 1). Das »u«, es steht für Unit (Geräteeinheit). Die Flags sind für die Ausgabe des RastPorts auf den Drukker nicht wichtig, wir übergeben daher eine 0. Der noch fehlende Parameter »io« ist die Adresse eines fertig initialisierten I/O-Datenblockes.

Die Initialisierung überlassen wir der Funktion CreateExtIO(). Sie verlangt neben der Größe des Datenblocks die Adresse eines MessagePorts als Parameter. CreatePort(name,pri) legt einen MessagePort an. Die Parameter hierbei sind ein Stringpointer auf den Namen des Ports und dessen Priorität.

Somit ist die Initialisierung des Datenblocks, in unserem Fall ist dies die Union PrIO, kein Problem mehr. Unsere Union erhält die drei Strukturen IOStdReq (für normale Textausgabe), IOPrtCmdReq (CommandRequest) und IODRPReq.

Die letzte Struktur ist für uns wohl am wichtigsten. Sie beschreibt den RastPort, der ausgegeben werden soll und die Art der Ausgabe. Listing 27 erklärt zusammen mit Tabelle 12 und 13 die wichtigsten Komponenten.

Nachdem in der Funktion HrdCpy() die notwendigen Werte übergeben sind, führt die Funktion DoIo() das unter io__Command angegebene Kommando aus. Printer-Device erlaubt mehrere Kommandos: CMD__WRITE und PRD__RAWWRITE zur Ausgabe von Texten, PRD__PTRCOMMAND zur Ausgaben von Kommandos und PRD__DUMPRPORT zur Ausgabe eines RastPorts.

Nach erfolgtem Ausdruck schließen wir alle Ressourcen und kehren zurück ins Hauptprogramm. HrdCpy() liefert TRUE (1), wenn der RastPort erfolgreich ausgedruckt werden konnte, im andern Falle FALSE (0).

Der Aufruf der Funktion ist denkbar einfach:

```
struct Screen *scr;
UWORD l,t,w,h;
USHORT s;
BOOL ok;
...
```

RastPort-Specials

Symbol	Wert	Bedeutung
SPECIAL__MILCOLS	0x001L	io__DestCols entsprechen 1/1000 inch
SPECIAL__MILROWS	0x002L	io__DestRows entsprechen 1/1000 inch
SPECIAL__FULLCOLS	0x004L	io__DestCols = maximal
SPECIAL__FULLROWS	0x008L	io__DestRows = maximal
SPECIAL__FRACCOLS	0x010L	Breite=maximal/io__DestCols
SPECIAL__FRACROWS	0x020L	Höhe =maximal/id__DestRows
SPECIAL__ASPECT	0x080L	Seitenverhältnisse ausgleichen
SPECIAL__DENSITY1	0x100L	Druckdichte minimal
SPECIAL__DENSITY2	0x200L	Druckdichte gering
SPECIAL__DENSITY3	0x300L	Druckdichte hoch
SPECIAL__DENSITY4	0x400L	Druckdichte maximal
SPECIAL__CENTER	0x040L	Ausdruck zentrieren

Tabelle 13. Die Spezial-Modi zum Ausdrucken von RastPorts

```
ok=HrdCpy(scr,l,t,w,h,s)
```

Dabei ist scr die Adresse des Screens, dessen RastPort Sie ausgeben möchten. Geben Sie NULL an, beziehen sich alle Daten auf den aktiven Screen. Die Variablen l,t,w und h beschreiben die Position der linken Seite, die der oberen Seite sowie Breite und Höhe des gewünschten Rechtecks im RastPort. Die Angaben beziehen sich auf die linke obere Ecke des Screens. Mit dem letzten Parameter legen Sie die Größe des Ausdrucks fest. Die Größe verhält sich proportional zu s. Ausnahme: den größten Ausdruck erhalten Sie bei der Angabe von s=0.



Bedienungsfreundlichkeit ist angesagt, auch bei selbstgeschriebenen Programmen. Vorbei sind die Zeiten, in denen ein Rechner den Bediener mit Befehlen, wie »Enter your choice!«, herumkommandierte – jetzt gibt es den Amiga, jetzt gibt es Gadgets und Pull-Down-Menüs.

Wir werden auf den folgenden Seiten ausführlich die Einsatzgebiete und die Programmierung dieser von Intuition bereitgestellten Bedienungselemente diskutieren. In diesem Rahmen erläutern wir, neben vielen Beispielprogrammen, weitere »Tool.h«-Routinen, die dem C-Programmierer die Arbeit erleichtern und den nötigen Aufwand auf ein Minimum reduzieren können.

Beginnen wir mit den Gadgets. Einige davon, nämlich die System-Gadgets, kennen Sie ja bereits. Sie haben sich am Anfang, bei der Besprechung der Windows, mit ihnen befaßt (Bild 2).

Intuition erlaubt neben diesen vordefinierten Gadgets auch die Benutzung eigener, anwendungsspezifischer Gadgets in beliebiger Form und Größe. Programme, wie zum Beispiel Preferences, die mit Gadgets nur so gespickt sind, erinnern eher an ein Schaltpult mit Reglern und Schaltern als an ein Softwareprodukt.

Gleichgültig, ob es sich nun um System-Gadgets handelt oder um solche, die man selbst erzeugt hat, immer hat der Anwender die Möglichkeit, Einstellungen bequem per Mausklick vorzunehmen.

Die Hauptaufgabe dieser Gadgets (laut Wörterbuch »Dingsda« oder »Apparat«) ist demnach die Weitergabe von Nachrichten oder Daten vom Anwender an ein Programm. Diese Nachrichten können Zahlenwerte, Texte oder einfach nur Boolesche (»ja/nein«-) Entscheidungen sein. Entsprechend dieser Aufgabengebiete unterscheidet man drei Gadget-Typen: Proportional-, Eingabe- und Bool-Gadgets.

Welches Gadget Sie einsetzen möchten, wo und in welcher Größe es auf dem Window erscheinen soll sowie weitere Einzelheiten teilen Sie Intuition – wie könnte es anders sein – per Gadget-Struktur mit.

Intuition verwaltet alle Gadgets eines Windows mittels einer Liste, bestehend aus verketteten Gadget-Strukturen. Die ersten Elemente dieser Liste sind immer die System-Gadgets.

Die (New)Window-Strukturkomponente FirstGadget verweist auf die Struktur des ersten User-Gadgets. Alle User-Gadgets sind mit dem jeweils ersten Strukturelement, dem Pointer NextGadget, verkettet (Bild 4). Der Pointer des letzten Gadgets erhält den Wert NULL.

Betrachten wir uns die Gadget-Struktur in Listing 28 etwas genauer. Die Variablen LeftEdge, TopEdge, Width und Height beschreiben, wie auch bei den Windows, Position und Dimension eines Gadgets. Die Position bezieht sich, solange kein GREL-Flag gesetzt ist, auf die linke obere Ecke des Gadgets, innerhalb des zugehörigen Windows.

Mittels »Flags« werden die Grundeigenschaften eines Gadgets festgelegt. Wir unterteilen die Gadget-Flags in drei Gruppen. Die erste Gruppe bestimmt, wie die Variablen LeftEdge, TopEdge, Width und Height zu deuten sind. Es handelt sich hier um die oben erwähnten GREL-Flags. Ist das Flag GRELBOTTOM gesetzt, so bezieht sich die Variable TopEdge nicht mehr auf die Oberkante des Windows, sondern auf dessen Unterkante. TopEdge muß daher in diesem Fall einen negativen Wert beinhalten.

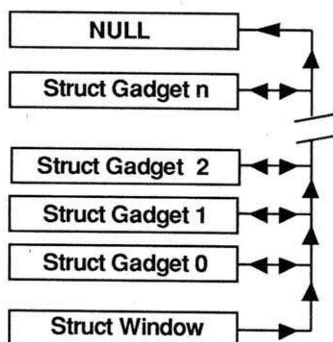


Bild 4. Die (New)Window-Struktur enthält das erste Glied der verketteten Gadget-Liste

Extras durch Gadget-Flags

Verändert der Anwender durch Vergrößern oder Verkleinern des Fensters die Position der Fensterunterkante, so verändert sich die Position des Gadgets relativ dazu.

```
struct Gadget
{
    struct Gadget *NextGadget;
    SHORT LeftEdge, TopEdge;
    SHORT Width, Height;
    USHORT Flags;
    USHORT Activation;
    USHORT GadgetType;
    APTR GadgetRender;
    APTR SelectRender;
    struct IntuiText *GadgetText;
    LONG MutualExclude;
    APTR SpecialInfo;
    USHORT GadgetID;
    APTR UserData;
};
```

Listing 28. Wie alle anderen Intuition-Elemente beschreibt man ein Gadget mittels einer Struktur

```
struct PropInfo
{
    USHORT Flags;
    USHORT HorizPot;
    USHORT VertPot;
    USHORT HorizBody;
    USHORT VertBody;
    USHORT CWidth;
    USHORT CHeight;
    USHORT HPotRes, VPotRes;
    USHORT LeftBorder;
    USHORT TopBorder;
};
```

Listing 29. Die kurze PropInfo-Struktur enthält alle Informationen über ein Proportional-Gadget

```
/*
\\    StrGad - Demo Text/Int-Gadget
*/
#include "TOOL.h"
#define ANZ_Z 20
#define BG    80
#define HG    9

struct Window *win;
UBYTE undo[ANZ_Z],text[ANZ_Z]="Text",zahl[ANZ_Z]="0",
      Ueberschrift[]="Text-Gadget   Int-Gadget";
SHORT pos[]={BG+1,HG+1, BG+1,1, BG,0, BG,HG, 0,HG, 1,HG+1, BG+1,HG+
1 };
struct Border border={      0,0,  2,0,  JAM1, 7, &pos[0],  NULL    };
struct StringInfo SInfo1={text,undo,0, ANZ_Z,0, 0,0,0,0,0,0,0,
NULL };
struct StringInfo SInfo2={zahl,undo,0, ANZ_Z,0, 0,0,0,0,0,0,0,
NULL };
struct IntuiText GText  ={2,0,JAM1,0,-15,NULL,&Ueberschrift[0],
NULL };

VOID error(Meldung)
    STRPTR Meldung;
{
    if(Meldung) printf("Kann %s nicht öffnen!?!\n",Meldung);
    if(win)     CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    struct Gadget SGad, IGad;     /*die beiden Gadgets
    String- und IntGadget */
    SHORT ok=0;
    USHORT id;
    ULONG class,idcmp=GADGETUP| MOUSEBUTTONS| CLOSEWINDOW,
          flags=WINDOWCLOSE| WINDOWDEPTH| WINDOWDRAG| WINDOWSIZING
          | ACTIVATE;
```

```
if(OpenLib()) error("Lib's");
if(!(win=GetWindow(NULL,0,0,230,70,idcmp,flags,"Text/
Int-Gadgets",NULL)))
    error("Window");
GetGadget(&SGad,win,10,40,BG,HG,GADGHCOMP,RELVERIFY|GADGIMMEDIATE,
    STRGADGET,&border,NULL,&GText,&SInfo1,1);/* String-
    Gadget mit TOOL.h */
GetGadget(&IGad,win,120,40,BG,HG,GADGHBOX,RELVERIFY|
GADGIMMEDIATE|LONGINT,
    STRGADGET,&border,NULL,NULL,&SInfo2,2);     /* Int-Gadget
    mit TOOL.h */
ActivateGadget(&SGad,win,NULL);                /* String-Gadget
aktivieren */
while(ok==0){        /* Bedingung ist nach CLOSEGADGET nicht mehr
erfüllt */
    Wait(1<<win->UserPort->mp_SigBit);
    class=GetMessage(win,0,&id,0);
    switch(class){
        case GADGETUP:{                 /* Returntaste betätigt */
            switch(id){
                case 1:{    /* Gadget mit ID Nummer 1: das
                String-Gadget */
                    printf("String: %s\n",SInfo1.Buffer);
                    ActivateGadget(&IGad,win,NULL); /*
                    Int-G  aktivieren */
                    break;
                }
                case 2:{        /* Gadget mit ID Nummer 2: das
                Int-Gadget */
                    printf("Zahl: %ld\n",SInfo2.LongInt);
                    ActivateGadget(&SGad,win,NULL);  /* Str-G
                    aktivieren */
                    break;
                }
            }
                break;
            }
            case CLOSEWINDOW:{
                ok++;
                break;
            }
        }
    }
    error(0);
}
```

Die Variable LeftEdge der Gadget-Struktur ist ein positiver Wert, der den Abstand zwischen der linken Fensterseite und linken Gadget-Seite beschreibt.

Diese ändert sich jedoch, wenn Sie das GRELRIGHT-Flag setzen. Jetzt interpretiert Intuition den in LeftEdge gespeicherten Wert als die Differenz zwischen der rechten Fensterseite und der linken Gadget-Seite. Geben Sie in diesem Fall für LeftEdge immer einen negativen Wert an.

Das Flag GRELWIDTH setzt den rechten Rand des Gadgets an die Position, die sich aus der Summe von LeftEdge und Width errechnet, bezogen auf den rechten Rand des Fensters (siehe auch Listing 29).

Das letzte Flag in dieser Gruppe, GRELHEIGHT, zeichnet den unteren Rand des Gadgets TopEdge+Height Punkte vom unteren Fensterrand entfernt.

Wir kommen zur zweiten Gruppe der Gadget-Flags. Hier legen wir die Reaktion auf einen Mausklick fest. Wir haben vier Möglichkeiten:

»GADGHCOMP« stellt das Gadget komplementärfarben dar.

»GADGHBOX« zeichnet im »JAM1:COMPLEMENT«-Modus einen Rahmen um das Gadget. Diese Möglichkeit nutzt man beispielsweise dann, wenn sich die Farbe des Gadgets auch beim Anwählen nicht verändern soll.

Listing 30 demonstriert den Unterschied zwischen GAD GHCOMP und GADGHBOX.

»GADGHIMAGE« zeichnet entsprechend der Gadget-Strukturkomponente Select-Render (siehe unten) einen neuen Rand oder aber ein völlig anderes Image.

Mit »GADGHNONE« verbieten Sie jede Veränderung des Gadgets bei der Anwahl mit der Maus. Achtung: Die Flags dieser Gruppe lassen sich nicht miteinander kombinieren.

Die letzte Gruppe der Gadget-Flags faßt alle Flags zusammen, die sich keiner der beiden anderen Gruppen zuordnen lassen. GADGIMAGE müssen Sie setzen, wenn auf dem Gadget ein eigenes Image zu sehen sein soll. Die Pointervariable GadgetRender (siehe unten) zeigt dann auf die Image-Struktur. Bei der Verwendung von Borders müssen Sie GAD-GIMAE nicht setzen.

»SELECTED« stellt ein Gadget im angewählten Zustand dar. Beachten Sie, daß ein String-Gadget mit gesetztem SELECTED-Flag nicht aktiv ist, es sieht nur so aus. Mit der Auto-Aktivierung von String-Gadgets befassen wir uns später.

Das Flag »GADGDISABLED« sorgt dafür, daß man das Gadget nicht anwählen kann. Dieser Zustand wird optisch durch eine Darstellung des Gadgets in Geisterschrift hervorgehoben. Den Zustand des Flags kann man mit den Funktionen

OnGadget(&gad,&win,&req)

und OffGadget(&gad,&win,&req) noch im nachhinein verändern. Leider zeichnet Intuition nach dem Aufruf von OnGadget() nur den Gadget-Rahmen oder, wenn definiert, das Image neu, löscht aber vorher den Gadget-Bereich nicht. Die Off-Darstellung bleibt daher zumindest teilweise sichtbar.

Die Parameter der beiden angesprochenen Funktionen sind gleich. Es handelt sich um Pointer auf die Strukturen des Gadgets, des Windows, dem das Gadget erscheinen soll, und des Requesters, falls es sich um ein Requester-Gadget handelt. Bei normalen Window-Gadgets übergibt man als dritten Parameter NULL.

Das nächste Element der Gadget-Struktur ist Activation. Sie können verschiedene Flags setzen und auch miteinander kombinieren. Wir besprechen sie am besten wieder nach Gruppen getrennt.

Mit Hilfe der ersten Gruppe können wir die »Activation« nach der Gadget-Anwahl einstellen.

Gadgets für jeden Zweck

»TOGGLESELECT« verwandelt unser Gadget in einen Flipflop-Schalter. Der Zustand des Gadgets kippt jeweils nach einem Mausklick um. Je nach Status erscheint es selektiert oder nicht selektiert. Der Anfangszustand ist »nicht selektiert«, was man allerdings mit dem oben beschriebenen Gadget-Flag SELECTED ändern kann.

»ENDGADGET« schließt bei der Anwahl eines Requester-Gadgets den Requester. Die folgenden drei Flags sind für die Gadget-Abfrage von Bedeutung.

»GADGIMMEDIATE« teilt dem Programm unverzüglich mit, wenn der Anwender das Gadget anwählt. Der Highlight-Zustand des Gadgets ist nur ganz kurz zu sehen.

Anders bei dem Flag »RELVERIFY«. Solange sich der Pointer, bei gedrückter Maustaste, über dem Gadget befindet, stellt sich dieses als angewählt dar. Und nur wenn er sich beim Loslassen der Maustaste noch über dem Gadget befand, gilt es als selektiert.

»FOLLOWMOUSE« setzt man, wenn man Informationen über die Mausbewegungen während der Gadget-Anwahl empfangen möchte. Sinnvoll ist dieses Flag zum Beispiel, wenn man die Werte eines Proportional-Gadgets (Schieberegler), noch während der Anwender es bedient, auswerten möchte.

Während zur Auswertung mit Hilfe der Flags GADGIMMEDIATE und RELVERIFY über den Message-Port die IDCMP-Flags GADGETDOWN und GADGETUP abgefragt werden, warten wir bei FOLLOWMOUSE-Gadgets auf MOUSEMOVE. Setzen Sie also entsprechend Ihrer Kontrollfunktion die richtigen IDCMP-Flags. Ein Beispiel für FOLLOWMOUSE-Gadgets finden Sie in Listing 29.

Gruppe 2 ordnet unser Gadget einem Fensterrand zu. Die Flags RIGHTBORDER, LEFTBORDER, TOPBORDER und BOTTOMBORDER finden besonders bei Proportional-Gadgets eine Anwendung. Ist eines der Flags gesetzt, »refresht« Intuition den Gadget-Rahmen oder das Image zusammen mit dem Window-Rahmen.

Gruppe 3 der Activation-Flags bezieht sich ausschließlich auf String-Gadgets. Normalerweise erscheint der Text in einem String-Gadget am rechten Rand des Eingabekästchens. Setzen Sie aber das Flag STRINGRIGHT, dann beginnt der Text am rechten Rand, und mit STRINGCENTER erscheint er stets in der Mitte des Feldes.

Möchten Sie Integerzahlen einlesen, setzen Sie einfach das Flag LONGINT. Bei der Eingabe sind dann nur noch Ziffertasten erlaubt. Alle anderen Tasten verursachen einen Bildschirmblitz.

Das letzte Activation-Flag, ALTKEYMAP, teilt Intuition mit,

daß Sie einen zweiten Zeichensatz verwenden möchten. Ein Pointer in der StringInfo-Struktur (siehe unten) verweist auf diesen Zeichensatz.

In der Gadget-Struktur geht es weiter mit GadgetType. Mit Hilfe dieser Flags legen wir den Gadget-Typ fest. Wir unterscheiden:
BOOLGADGET – das Ja/Nein-Gadget,
STRGADGET – dient zur Eingabe von Zahlen oder Zeichenketten und
PROPGADGET – die ein- oder zweidimensionalen Schieberegler.

Zwei weitere Flags bestimmen das Ausgabeelement des Gadgets. Setzen Sie das GZZGADGET-Flag, wenn Sie ein Gimmezerozero-Window verwenden. Möchten Sie ein Gadget in einem Requester plazieren, so teilen Sie dies Intuition über das REQGADGET-Flag mit.

Die beiden nächsten Komponenten sind die Pointer GadgetRender und SelectRender. Hier kann ein Zeiger auf eine Border-Struktur oder auf eine Image-Struktur stehen.

GadgetRender ist zu sehen, wenn das Gadget nicht angewählt ist. Nach einem Mausklick über dem Gadget wird SelectRender sichtbar. Verwendet man GadgetRender, so muß GADGIMAGE unter »Flags« gesetzt sein. Setzen Sie GADGHIMAGE, so beachtet Intuition den Pointer unter SelectRender.

Bei der Verwendung von Proportional-Gadgets müssen Sie, auch wenn Sie kein eigenes Image definiert haben, GadgetRender mit einem Image-Strukturpointer belegen. Intuition legt dort die Daten für den vom System zur Verfügung gestellten »Schiebeknopf« ab.

Beachten Sie bitte, daß sich Ihre selbstgeschaffenen Rahmen oder Images nicht selbständig einem größenveränderlichen GREL-Gadget anpassen. Es folgt in der Gadget-Struktur der Pointer GadgetText. Es handelt sich hierbei um die Adresse einer IntuiText-Struktur, wir haben sie bereits besprochen. Tragen Sie hier bitte NULL ein, wenn Sie Ihr Gadget nicht beschriften möchten.

In MutualExclude, dem nächsten Strukturelement, kann beschrieben werden, wie einzelne Gadgets zusammenarbeiten. Intuition ignoriert es jedoch in der vorliegenden Betriebssystemversion.

Der Pointer SpecialInfo verlangt die Adresse einer PropInfo-Struktur, wenn Sie mit Proportional-Gadgets arbeiten, die Adresse einer StringInfo-Struktur bei String-Gadgets oder aber NULL bei Boolean-Gadgets.

Wie wir später noch sehen werden, liefert der IDCMP zwar Informationen über die Anwahl eines Gadgets, nicht aber darüber, um welches es sich dabei handelt. Zur Unterscheidung beim Arbeiten mit mehreren Gadgets vereinbart man in der Gadget-Struktur eine ganzzahlige positive Identitätsnummer: die »GadgetID«. Unsere Tool.h-Funktion GetMessage() ermittelt die ID auf Wunsch automatisch.

Intuition ignoriert diese Variable, denn dieses und auch das letzte Element der Gadget-Struktur namens UserData ist nur für den Programmierer gedacht. UserData ist ein Zeiger auf eigene Daten, die im Zusammenhang mit dem Gadget stehen.

ID zur Entscheidung

Bild 4 zeigt schematisch die statische Initialisierung einer Anzahl Gadgets. Da das jeweils folgende Gadget (NextGadget) bei der Initialisierung bekannt sein muß, baut sich die Liste von unten nach oben auf.

In der NewWindow-Struktur tragen wir die Adresse der ersten Gadget-Struktur ein.

Nach einem erfolgreichen Aufruf der Funktion OpenWindow() befinden sich dann alle Gadgets auf den vereinbarten Positionen.

Falls Teile eines Gadgets überschrieben oder überzeichnet wurden, zeichnet RefreshGadgets(&gad,&win,&req) alle Gadgets einer Liste – ab dem als ersten Parameter genannten – neu. Die beiden anderen Parameter zeigen, wie bei den meisten anderen Gadget-Funktionen, auf die Strukturen des Ausgabewindows und, falls erwünscht, des Ausgabe-Requesters.

Eine ähnliche, jedoch erweiterte Funktion ist RefreshGList(&gad,&win,&req,n). Sie erlaubt ein »Refreshen« von n Gadgets, ab dem als ersten Parameter genannten. Bei der Angabe von n=–1, zeichnet Intuition die ganze Gadget-Liste neu.

Da wir für jedes Gadget eine eigene Struktur anlegen und diese auch mit Werten versorgen müssen, bläht sich der Quellcode unserer Programme bei der Verwendung mehrerer Gadgets gewaltig auf. Dies trägt natürlich nicht gerade zu deren Übersichtlichkeit und Nachvollziehbarkeit bei. Eine Vereinfachung muß her – eine Aufgabe für Tool.h.

Die Gadget-Funktion GetGadget() (siehe Listing 1) aktiviert auf einem bereits bestehenden Window ein beliebiges Gadget. Der Aufruf verlangt viele Parameter: GetGadget(&g, &w,l,t,w,h,f,a,ty,gr,sr,&tx,&si,id). Da Sie ja nun die Gadget-Struktur kennen, bereitet diese Liste für Sie keine Probleme. Die Parameter sind:
»&g« ist der Zeiger auf eine Gadget-Struktur. Tool.h initialisiert diese, so daß Sie sie beim Aufruf anderer Gadget-Funktionen einsetzen können.
»&w« ist die Adresse des Fensters, auf dem das Gadget erscheinen soll;
»l/t/w/h« beschreiben Position und Dimension des Gadgets;
»f« sind die Flags; in »a« werden die Activation-Flags übergeben;
»ty« ist der Gadgettyp;
»gr« steht für GadgetRender;
»sr« bedeutet SelectRender;
»&tx« ist die Adresse der IntuiText-Struktur;
»&si« ist ein Zeiger auf SpecialInfo und »id« ist die Nummer des Gadgets.

GetGadget() kehrt mit der Nummer der Position, an der das Gadget in die Gadget-Liste eingefügt wurde, zurück.

Nachdem in dieser Funktion die einzelnen Parameter den richtigen Strukturkomponenten zugewiesen sind, rufen wir die Funktion AddGadget(&win, &gad,&req,nr) auf. Abgesehen von »nr« kennen Sie die Parameter bereits, es sind die gleichen wie zum Beispiel bei RefreshGadgets(). Freundlicherweise sind jedoch die beiden ersten Parameter gegenüber den meisten anderen Gadget-Funktionen vertauscht, was gelegentlich zu Schreibfehlern führen kann.

»AddGadget()« fügt einer bestehenden Gadget-Liste ein neues Gadget, an der Position »nr«, ein. Möchte man das Gadget der Liste anhängen, so gibt man für n den Wert –1 an. Diese Möglichkeit nutzen wir in Tool.h.

Ja oder nein: Bool-Gadgets

Der Returnwert der Funktion entspricht der tatsächlichen Position des Gadgets in der Liste. Wir reichen diesen Wert nach dem Aufruf von RefreshGadget() weiter. Ein Aufruf wie

```
nr=GetGadget(...);
```

aktiviert ein Gadget und liefert dessen Nummer in der Gadget-Liste. Denken Sie aber daran, daß die System-Gadgets vor den userdefinierten gelagert sind. Der Returnwert ist entsprechend höher.

Die Boolean-Gadgets sind wohl die einfachsten aber auch die am meisten eingesetzten Intuition-Gadgets. Sie sind immer dann gefragt, wenn ein Programm nach einer Ja/Nein-Entscheidung verlangt oder die Auswahl von verschiedenen Möglichkeiten bietet.

Man findet sie in jedem einigermaßen komfortablen Programm. So wählt man beispielsweise in Textprogrammen über Bool-Gadgets den Zeilenabstand oder die Schriftart, in Grafikprogrammen stellt man bequem Pinselbreite und Zeichenmodus ein. Die Programmierung selbst ist einfach, schauen wir uns Listing 31 an. SetStyle aktiviert sieben Bool-Gadgets am rechten Rand des CLI-Fensters. Dieser Bereich ist sehr gut geeignet, da er wegen der SIZEBRIGHT-Einstellung (siehe unter Windows) nicht hochscrollt, wenn der Cursor die unterste CLI-Zeile erreicht.

Jedes der Gadgets schaltet, wenn selektiert, einen anderen SoftStyle-Zeichensatz ein. Die sieben Gadget-Strukturen vereinbaren wir in Listing 31 gleich zu Beginn der Hauptfunktion main().

Nach dem Öffnen der Libraries verändert der Aufruf der Funktion ModifyIDCMP() die IDCMP-Flags des gerade aktiven CLI-Fensters. Wie Sie wissen, sendet uns Intuition über den MessagePort nur solche Nachrichten, die wir durch Kombination der IDCMP-Flags anfordern.

Für die Gadget-Kontrolle betrachten wir zunächst zwei Flags: GADGETDOWN und GADGETUP. Zu MOUSEMOVE im Zusammenhang mit Gadgets erfahren Sie später mehr.

Intuition sendet uns entsprechend der gesetzten IDCMP-Flags eine Nachricht, wenn der Anwender die linke Maustaste über dem Gadget drückt (DOWN) oder losläßt (UP). Möchte man in jedem Fall eine Nachricht, so kann man die Flags auch kombinieren, also mit ODER verknüpfen:

```
GADGETDOWN|GADGETUP
```

Der nächste Schritt in SetStyle.c ist der Aufruf der neuen Gadget-Funktion. Er erfolgt in einer Schleife siebenmal. Die Y-Position des Gadgets und die Gadget-Kennummer sind ab-

```
/*
\\    SetStyle - BOOLGADGET-Demo
*/
#include "TOOL.h"
#define ANZ_G 7
#define HOEHE 238/ANZ_G
#define BREITE 15

SHORT pos[]={    0,0, BREITE,0, BREITE,HOEHE, 0,HOEHE, 0,
     0 }; /* Koordinaten */
struct Border border={   -1,-1, 1,0, JAM1, 5,&pos[0], NULL };
     /* Border */

VOID main()
{
    struct Window *win;
    struct Gadget gad[ANZ_G];
    ULONG class;
    USHORT id;
    SHORT i,style,ende=0;

    if(OpenLib()){
        printf("Kann Lib's nicht öffnen!\n");
        exit(0);
    }
    ModifyIDCMP(win=ACTIVE_WINDOW,GADGETUP);          /* IDCMP-Flags
    ändern */
    for(i=0;i<ANZ_G;i++){
        GetGadget(&gad[i],win, 640-BREITE,10+i*HOEHE,BREITE,HOEHE,
        GADGHCOMP,
                RELVERIFY| GADGIMMEDIATE, BOOLGADGET,&border,
                NULL,NULL,NULL,i);
    }
    printf("\nENDE\nNormal\nFett\nSchwarz\nSchräg\nUnterstrichen\
    nRevers\n");
    while(ende==0){
        Wait(1<<win->UserPort->mp_SigBit);
        if((class=GetMessage(win,0,&id,0))==GADGETUP){
            if((style=(id<6) ? id-1 : id+1) < 0) ende++;
            else printf("\033[%dmAa Bb Cc\n",style);
        }
    }
    for(i=0;i<ANZ_G;i++) RemoveGadget(win,&gad[i]);     /* Gadgets
    streichen */
    ModifyIDCMP(win,0);                          /* IDCMP-Flag
    für CLI-Fenster */
    RefreshWindowFrame(win); /* Windowrahmen neuzeichnen und Gadgets
    löschen */
    CloseLib();
}
```

Listing 31. Umschalten der SoftStyle-Zeichensätze mit Boolean-Gadgets

hängig vom Schleifenzähler. Als Flag wählen wir GADGHCOMP, das bedeutet Komplementär-Darstellung bei der Anwahl. Die Activation-Flags RELVERIFY und GADGIMMEDIATE sorgen für die direkte Benachrichtigung des Programms. Außerdem soll das Gadget nur als angewählt gelten, wenn sich der Mauszeiger beim Loslassen der Maustaste noch über dem Gadget befand.

Ein Zeiger auf die statisch initialisierte Borderstruktur sorgt für eine Umrahmung der einzelnen Gadgets. Die Positionsangaben innerhalb dieser Struktur beziehen sich auf das Gadget. Der Parameter SpecialInfo ist bei Bool-Gadgets immer NULL. Der letzte Parameter, die Identitätsnummer, ist identisch mit dem Schleifenzähler.

Mit der folgenden printf()-Funktion geben wir die Bedeutung der einzelnen Gadgets auf dem Bildschirm aus. Die Initialisierung von sieben IntuiText-Strukturen hätte den Rahmen eines Bool-Gadget-Demos gesprengt.

Aber vielleicht versuchen Sie selbst einmal jedes Gadget mit einem Buchstaben zu kennzeichnen, um so auf dessen Funktion hinzuweisen?

Eine while-Schleife schließt sich an. Mit der ersten Zeile des Schleifenrumpfes wartet das Programm, bis eine Intuition-Nachricht anliegt. SetStyle belastet das System also kaum, so daß Sie es auch mit »RUN SetStyle« starten und weiter im CLI arbeiten können.

Die Tool.h-Funktion GetMessage() empfängt die Nachricht. Ist diese gleich GADGETUP, so lesen wir die Kennummer des gewählten Gadgets und wandeln sie in die möglichen SoftStyle-Werte um – dies sind die Werte 0 bis 4 und 7.

Wie erhalten wir die Gadget-ID? Wir haben es schon einmal bei der Besprechung der Tool.h-Funktion GetMessage() angedeutet. Gibt man dieser als dritten Parameter die Adresse einer USHORT-Variablen mit, belegt die Funktion die entsprechende Variable mit dem Wert der Gadget-ID des gewählten Gadgets.

```
USHORT id;
...
class=GetMessage(win,0,
&id,0);
```

Eine Komponente innerhalb der von GetMessage() empfangenen IntuiMessage-Struktur verweist auf das Objekt, das die Nachricht betrifft: IAddress. In unserem Fall zeigt IAddress auf die Struktur des gewählten Gadgets, wo wir auch die benötigte Nummer finden. Als nächstes geben wir ein paar Zeichen des neu eingeschalteten Softzeichensatzes aus. Die Umschaltung erfolgt mit einer »Console Control Sequence«, zum Beispiel

```
printf("\ 033[0m");
```

für Normalschrift. Das oberste Gadget hat die ID 0. Wird es selektiert, erhält die Variable »style« in unserem Programm den Wert -1. Die Bedingung der while-Schleife ist nicht mehr erfüllt, es folgt das Programmende.

Komfortabel: String-Gadgets

Zuvor streicht jedoch RemoveGadget(&win,&gad) die Gadgets aus der Liste. Die Parameter sind Pointer auf Window- und Gadget-Struktur. Vorsicht, RemoveGadget() löscht die Gadgets aus der Liste, aber nicht vom Bildschirm. In SetStyle.c erreichen wir dies einfach mit RefreshWindowFrame().

Nachdem die IDCMP-Flags wieder auf null gesetzt sind, schließen wir mit der Tool.h-Funktion CloseLib() die benötigten Libraries.

Intuition bietet zwei Arten von String-Gadgets an: Text- und Inter-Gadgets. Sie ermöglichen die Eingabe von Zeichenketten beziehungsweise ganzen Zahlen über die Tastatur.

Die Eingabe von Float-Zahlen ist nicht direkt implementiert. Es ist aber denkbar, diese mittels Textgadget als Zeichenkette einzulesen, um sie dann zum Beispiel über die systemunabhängige Funktion atof() umzuwandeln. String-Gadgets werden in allen möglichen Arten von Programmen eingesetzt, zum Beispiel zur Eingabe von Filenamen oder Einstellparametern. Das Textgadget der Rename-Routine der Workbench hat sicher jeder schon einmal bedient.

Die Eingabe ist komfortabel, dem Bediener stehen einige Editiertasten zur Verfügung: Die Cursortasten links und rechts bewegen den Cursor in die entsprechende Richtung. Kombiniert man diese Tasten mit Shift, springt der Cursor an den Anfang oder an das Ende des Eingabestrings. Die DEL-Taste löscht das Zeichen unter, die BACKSPACE-Taste das Zeichen links vom Cursor. Die RETURN-Taste schließt die Eingabe ab. Wenn das RELVERIFY-Flag gesetzt ist, sendet Intuition eine Nachricht.

Die Kombination <rechte Amiga-Taste x> löscht die gesamte Eingabe, <rechte Amiga-Taste q> ist die zugehörige »Undo«-Funktion.

Im Grunde unterscheidet sich die Programmierung eines String-Gadgets nur in wenigen Punkten von der eines Boolean-Gadgets. Dies ist ein großer Vorteil für uns, denn wir können wieder unser Tool.h einsetzen und so Platz und Arbeit sparen.

Ein wichtiges Unterscheidungsmerkmal ist natürlich die Angabe von STRGADGET als Gadget-Type. Weiter belegen wir hier erstmals die Komponente SpecialInfo der Gadget-Struktur. Dort erwartet jetzt Intuition die Adresse einer StringInfo-Struktur (Listing 32).

Diese Struktur ist schnell erklärt, denn die meisten Variablen initialisiert Intuition selbst.

»Buffer« ist ein Zeiger auf einen Puffer, der groß genug sein muß, den mit Null abgeschlos-

```
struct StringInfo
{
    UBYTE *Buffer;
    UBYTE *UndoBuffer;
    SHORT BufferPos;
    SHORT MaxChars;
    SHORT DispPos;
    SHORT UndoPos;
    SHORT NumChars;
    SHORT DispCount;
    SHORT CLeft, CTop;
    struct Layer *LayerPtr;
    LONG LongInt;
    struct KeyMap *AltKeyMap;
};
```

Listing 32. Für die Verwendung eines String-Gadgets fordert Intuition die Initialisierung einer StringInfo-Struktur

senen Eingabestring aufzunehmen. »UndoBuffer« ist der Puffer für die Undo-Funktion. Er muß mindestens die Größe von »Buffer« haben. Findet Intuition hier eine 0, so funktioniert die Undo-Funktion nicht.

Hat man mehrere String-Gadgets definiert, kommt man oft dennoch mit einem einzigen Undo-Puffer aus. Geben Sie in StringInfo einfach immer die gleiche Adresse unter Undo-Puffer an.

»BuffPos« ist die Cursorposition im Eingabefeld.

»MaxChars« gibt die maximale Anzahl der Eingabezeichen plus eins, wegen der abschließenden Null, an.

»DispPos« gibt an, welches Zeichen im Puffer an erster Stelle im Eingabefeld dargestellt werden soll.

Die folgenden sieben Elemente verwaltet Intuition selbst, wir geben bei der Initialisierung einfach eine 0 an.

»UndoPos« beschreibt die Cursorposition im Undo-Puffer.

»NumChars« ist die Anzahl der Zeichen im Puffer.

»CLeft« und »CTop« ist die Position der linken oberen Ecke des Eingabefeldes im Gadget.

»LayerPtr« zeigt auf die zugehörige Layer-Struktur.

»LongInt« enthält den Eingabewert eines Integer-Gadgets.

Den Inhalt des Zeigers »AltKeyMap« geben wir wieder selbst an. Es ist ein Zeiger auf eine eigene Tastaturtabelle, in diesem Fall muß das Activationflag ALTKEYMAP gesetzt sein.

Genug erzählt! Schauen wir uns die Programmierung lieber an einem praktischen Beispiel an. Das erklärt mehr als 1000 Worte:

Listing 32 öffnet ein Fenster mit einem Text- und einem Integer-Gadget. Nach dem Start gibt das Programm jede mit <RETURN> abgeschlossene Gadget-Eingabe im CLI-Fenster aus. Die Gadgets aktivieren sich dabei wechselseitig, das Text-Gadget ist sofort nach dem Programmstart aktiv. Ohne diese Selbstaktivierung müßte der Programmbediener jedes Gadget vor der Eingabe mit der Maus anwählen. Zu Beginn des Listings 32 werden notwendige Strukturen wie die Border-, die StringInfo- und die IntuiText-Struktur initialisiert.

Innerhalb von main() vereinbaren wir gleich zwei Gadget-Strukturen. Die erste für das Text-Gadget, die andere für das Integer-Gadget. Nach dem Öffnen der Libraries und des Windows startet ein Aufruf der neuen Tool.h-Funktion GetGadget() die beiden Gadgets.

Das Textgadget ist mit einem IntuiText versehen. Beide Gadgets nutzen die gleiche Borderstruktur und auch – vergleichen Sie bitte die beiden StringInfo-Strukturen – den gleichen Undo-Puffer.

Als Activation-Flags sind bei beiden Gadgets RELVERIFY und GADGIMMEDIATE gesetzt. Das Integer-Gadget benötigt noch zusätzlich LONGINT. Die Gadget-Flags sind unterschiedlich. Während beim Textgadget GADGHCOMP für eine komplementäre Darstellung sorgt, stellt GADGHBOX innerhalb des Integer-Gadgets den Cursor als Rahmen oder Box dar.

Die Funktion ActivateGadget(&gad,&win,&req) aktiviert das erste Gadget. Die Parameter entsprechen denen der Ihnen schon bekannten Gadget-Funktionen. In der folgenden while-Schleife analysieren wir die von Intuition gelieferten Nachrichten. Im Falle CLOSEWINDOW erfolgt das Programmende. Empfangen wir GADGETUP, so entscheiden die GadgetIDs, welcher case-Zweig ausgeführt wird.

Im ersten Fall gibt eine printf()-Funktion die Zeichenkette aus:

```
printf("String: %s\n" ,
SInfo1.Buffer);
```

Hat der Bediener das Integer-Gadget benutzt, geben wir die ins Gadget eingetragene Zahl aus:

```
printf("Zahl: %ld\n",
SInfo2.LongInt);
```

In jedem Fall aktiviert ActivateGadget() das jeweils andere Gadget. Auf die Einrichtung eines default-Zweigs innerhalb der beiden switch-Anweisungen haben wir bewußt verzichtet.

Elegant: Proportional-Gadgets

Mit Hilfe dieser leistungsfähigen Intuition-Eingabeelemente in Form eines Schiebereglers kann man während des Programmablaufs Werte zwischen zwei Grenzen beliebig verändern.

Man »ergreift« mit der Maus den Schiebeknopf und bewegt ihn in eine andere Position. Dabei ändert sich ein bestimmter Wert kontinuierlich wie bei einem Potentiometer.

Proportional-Gadgets findet man in allen Arten von Programmen, zur Einstellung von Geschwindigkeiten, Positionen oder der Bildschirmfarben. Die in den Workbench-Windows installierten Rollbalken sind nichts anderes als Proportional-Gadgets.

Zur Verwaltung eines Proportional-Gadgets ist, wie Sie richtig befürchten, eine weitere Struktur nötig: die «PropInfo«-Struktur. Weiter oben haben wir bereits angedeutet, daß Intuition deren Adresse in der Gadget-Struktur unter SpecialInfo erwartet.

Interessant in dieser Struktur sind eigentlich nur die ersten fünf Variablen, alle anderen setzt Intuition selbst:

Flags bestimmen unter anderem über das Aussehen des Gadgets. Es existieren fünf Flags, eines davon, »KNOBHIT«, setzt Intuition, wenn der Anwender den Schiebeknopf anklickt. Das Flag bleibt so lange gesetzt, bis Intuition einen Mausklick außerhalb des Knop-

```
/*
\\    Prop.c - demonstriert Proportional-Gadgets
*/
#include "TOOL.h"
#define GS (0xFFFF/32)                  /* Gadget mit 32 '
Schieberstellungen' */
#define VP &ACTIVE_SCREEN->ViewPort                       /* Zeiger
auf ViewPort */

struct Window *win;
struct Image image;
struct PropInfo PInfo={
    FREEHORIZ| AUTOKNOB, GS*16,0, GS,0, 0,0,0,0,0,0
};
VOID error(Meldung)
    STRPTR Meldung;
{
    if(Meldung) printf("Kann %s nicht öffnen!?!\n",Meldung);
    if(win)     CloseWindow(win);
    CloseLib();
    exit(0);
}

VOID main()
{
    struct Gadget gad;

    ULONG class=0,idcmp=MOUSEMOVE| CLOSEWINDOW,
          flags=WINDOWCLOSE| WINDOWDEPTH| WINDOWDRAG| WINDOWSIZING
          | ACTIVATE;
    SHORT i, ok=0, r[4],g[4],b[4], wert, rot,gruen,blau;

    if(OpenLib()) error("Lib's");
    if(!(win=GetWindow(NULL,0,0,160,60,idcmp,flags,"Re-Darker",
    NULL)))
        error("Window");
    win->MinWidth=win->MinHeight=60;
    GetGadget(&gad,win,10,20,-20,-35,GRELWIDTH| GRELHEIGHT,
    FOLLOWMOUSE,
                                  PROPGADGET,&image,NULL,NULL,&
                                  PInfo,0);
    for(i=0;i<4;i++)  GetCol(VP,i,&r[i],&g[i],&b[i]);
    while(class!=CLOSEWINDOW){
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage(win,0,0,0);
        if(class==MOUSEMOVE){                    /* Das Gadget wurde
        angewählt */
            wert=PInfo.HorizPot/GS-16;             /* neuer
            Wert eingestellt */
            for(i=0;i<4;i++){             /* die 4 Farbregister des
            WB-Screens */
                rot  =(((r[i]-wert)<0)?0:r[i]-wert);
                if(rot>15) rot=15;
                gruen=(((g[i]-wert)<0)?0:g[i]-wert);
                if(gruen>15) gruen=15;
                blau =(((b[i]-wert)<0)?0:b[i]-wert);
                if(blau>15) blau=15;
                SetRGB4(VP,i,rot,gruen,blau);          /* neue
                Farben setzen */
            }
            while((class=GetMessage(win,0,0,0))==MOUSEMOVE);
        }
    }
    error(0);
}
```

Listing 33. Re-Darker: ein Proportional-Gadget als Helligkeitsregler

fes registriert. Um die Umrahmung eines Proportional-Gadgets brauchen Sie sich nicht zu kümmern, Intuition zeichnet automatisch einen passenden Rand. Dies können Sie allerdings, wenn nötig, mit dem »PROPBORDERLESS«-Flag verbieten. Intuition nimmt Ihnen auch die Definition des Schiebeknopfes ab. Falls Sie das wünschen, setzen Sie »AUTOKNOB« und tragen Sie unter GadgetRender in der Gadget-Struktur einen Zeiger auf eine Image-Struktur ein. Die Größe des Intuition-Knopfes ist von den Maßen des Containers, das ist der Bereich innerhalb des Rahmens, und der Anzahl der möglichen Werte, also der Schrittweite, abhängig. Wollen Sie einen eigenen GadgetKnopf definieren, legen Sie in GadgetRender die Adresse der von Ihnen initialisierten Image-Struktur ab.

»Props« unter der Lupe

Die beiden restlichen Flags bestimmen über die Bewegungsrichtung des Gadgets. Man kann den Knopf in der Horizontalen bewegen, wenn »FREEHORIZ« gesetzt ist. »FREEVERT« erlaubt die Bewegung in vertikaler Richtung. Kombinieren Sie die beiden Flags, so ist der Knopf in beide Richtungen beweglich. Ein solches zweidimensionales Proportional-Gadget finden Sie in Listing 42, Sie können damit die Position des Bildes auf dem Monitor beliebig einstellen.

HorizPot ist die horizontale Position des Knopfes beim Erscheinen des Gadgets auf dem Bildschirm in Prozent, bezogen auf MAXPOT (= 0xFFFF). Weiter unten legen wir in HorizBody die Schrittweite und damit die Anzahl der Schieberstellungen fest. Der Wert HorizPot ist nun einfach die gewünschte Start-Schieberstellung, multipliziert mit dem unter HorizBody gespeicherten Wert.

Sie müssen HorizPot nur dann initialisieren, wenn das PropInfo-Flag FREEHORIZ gesetzt ist.

VertPot beinhaltet bei Gadgets mit gesetztem FREEVERT-Flag die vertikale Position des Schiebers beim Erscheinen des Gadgets auf dem Bildschirm. Hier erreichen Sie die Grundeinstellung, indem Sie einen Ausgangswert mit dem Inhalt der Variablen VertBody multiplizieren.

In **HorizBody** und **VertBody** tragen Sie die Schrittweite einer Schieberstellung ein. Sie errechnet sich, indem man einfach den durch die Konstante MAXBODY symbolisierten Wert 0xFFFF durch die Anzahl der benötigten Schieberstellungen dividiert.

Die restlichen Elemente der PropInfo-Struktur werden zwar von Intuition gesetzt, wir wollen sie aber dennoch kurz besprechen:

CWidth und **CHeight** sind die Breite und die Höhe des Containers.

HPotRes und **VPotRes** beschreiben das Inkrement des Knopfes pro Schieberstellung, es berechnet sich aus MAXBODY/(Schieberstellungen-1).

LeftBorder und **TopBorder** geben die Position der linken oberen Ecke des Gadgets an. Mit der Funktion ModifyProp (&gad,&win,&req,Flags,HorizPot,VertPot,HorizBody,VertBody) ist es möglich, die ersten fünf Variablen der PropInfo-Struktur jederzeit zu verändern. ModifyProp() »refresht« das Gadget nach der Modifizierung selbständig.

Die ersten drei Parameter sind im Zusammenhang mit Gadget-Funktionen bereits mehrfach erklärt worden, die restlichen fünf sind identisch mit den gerade besprochenen Komponenten der PropInfo-Struktur.

Die Abfrage der Proportional-Gadgets mit GADGETDOWN und GADGETUP funktioniert analog der Abfrage von Boolean- und String-Gadgets.

Die unmittelbare Abfrage und die Auswertung der Gadgets lernen wir bei der folgenden Besprechung des Beispielprogramms Prop.c (Listing 33) kennen.

Das übersetzte Programm öffnet ein Fenster mit einem Schieber zum Einstellen der Bildschirmhelligkeit. Die Veränderung der Helligkeit erreichen wir durch Inkrementierung oder Dekrementierung der vier Farben des Workbenchscreens.

Nach der Einbindung des Headerfiles und der Festlegung einiger Konstanten vereinbaren wir unter anderem einen Zeiger auf eine Image-Struktur. Diese Struktur wird später die Werte des von Intuition gestellten Schiebeknopfes beinhalten.

Die statische Initialisierung der PropInfo-Struktur folgt. Sie verrät, daß in dem zugehörigen Gadget ein Intuitionknopf in 32 Schritten waagerecht verschiebbar ist und daß sich dieser Kopf zu Beginn an der 16. Position, also in der Mitte, befindet. Die Vereinbarung der Gadget-Struktur finden Sie am Anfang der main()-Funktion. Da wir dieses direkt, also noch während der Bedienung, abfragen möchten, setzen wir als IDCMP-Flag für das Window neben CLOSEWINDOW noch MOUSEMOVE. Intuition sendet uns diese Nachricht, wenn der Anwender ein Gadget mit gesetztem FOLLOWMOUSE-Activationflag angewählt hat.

Nach dem Öffnen der Libraries und des Windows legen wir die Mindestausdehnung für das gerade geöffnete Fenster fest. Es folgt die Aktivierung des, wie Sie aus den gesetzten Flags ersehen, größenveränderlichen Gadgets.

Die Tool.h-Funktion GetCol(), in der folgenden for-Schleife, liest die Farbwerte des aktiven Screens aus und splittet sie in ihre Rot-Grün-Blau-Werte. In einer while-Schleife überwachen wir den Messageport. Wir erwarten die Werte CLOSEWINDOW und MOUSEMOVE. Im Falle CLOSEWINDOW beendet die Funktion error() das Programm.

Wählt der Anwender das Gadget an, so sendet Intuition MOUSEMOVE. Die PropInfo-Variable HorizPot beinhaltet den momentan eingestellten Wert:

```
wert=PInfo.HorizPot/GS;
```

»GS« ist dabei die Konstante, die die Anzahl der Schieberstellungen festlegt, in unserem Beispiel bedeutet GS (0xFFFF/32). Im Beispielprogramm subtrahieren wir noch 16, so daß der Wert der Variablenwert negativ, positiv oder null sein kann.

Die Berechnung der neuen Werte der vier Workbenchfarben ist recht einfach: Wir müssen nur von den oben ermittelten Farbwerten unseren Gadget-Wert subtrahieren.

Je nach Größe und Vorzeichen der Variablen wert erscheint der Bildschirm nach dem Aufruf der Funktion SetRGB4() heller oder dunkler.

Sie sind es sicher schon gewohnt, am Ende eines jeden Kapitels eine Reihe von Problemstellungen vorzufinden. Diesmal ist es nur ein einziges, allerdings deckt es den besprochenen Stoff voll ab und ist entsprechend umfangreich:

- Öffnen Sie ein Window mit einem Boolean-, einem Integer- und drei Proportional-Gadgets. In das Integer-Gadget soll der Anwender Ihres Programms eine Farbregisternummer eintragen können. Der Inhalt dieses Registers, die Farbe also, erscheint nach einem <RETURN> auf dem Boolean-Gadget. Diese Farbe soll über die drei Schieberegler einstellbar sein. Der Farbwert wird aber erst in das entsprechende Register übernommen, wenn der Anwender seine Einstellung mit einem Klick über dem Boolean-Gadget bestätigt.



Sie sehen, mit Gadgets kann man richtige Menü-Felder aufbauen. Eine andere, sehr komfortable Art der Darstellung mehrerer Auswahlmöglichkeiten, ist das bekannte Intuition Pull-Down-Menü. Der Einsatz dieser Menüs ist für Anwender und Programmierer gleichermaßen interessant. Für den einen bedeuten sie höchsten Bedienungskomfort, für den anderen eine Entlastung durch bereits existierende Betriebssystemroutinen. Wir zeigen Ihnen auf den folgenden Seiten, wie man dieses Bedienungselement in eigenen Programmen nutzt.

Damit werden unsere Programme noch komfortabler und damit auch professioneller. Wer als Amiga-Programmierer seine Produkte anwenderfreundlich gestalten möchte, kommt an Pulldown-Menüs einfach nicht vorbei.

Der Einsatz eines Menüs bietet sich immer da an, wo der Anwender aus einer Anzahl verschiedener Programmpunkte einen auswählen und so über den weiteren Programmverlauf entscheiden kann.

Natürlich könnte man ein Menü auch, wie im Beispiel Palette, aus Gadgets aufbauen oder einfach nur eine Ziffer über die Tastatur einlesen und entsprechend im Programm verzweigen. Die genannten Möglichkeiten setzen allerdings ein eigenes Menüfeld, auf dem alle Möglichkeiten dargestellt sind, voraus. Hier zeigt sich bereits der erste Vorteil der Intuition-

menütechnik:

Pull-Down-Menüs sind platzsparend in der Screen-Titelleiste untergebracht. Erst beim Betätigen der rechten Maustaste - des sogenannten »Menubuttons« - erscheinen dort die Titel der Hauptmenüs. Bewegt man den Mauszeiger mit gedrückter rechter Maustaste über einen solchen Menütitel, klappt das Menü nach unten.

Jetzt sind die einzelnen Menüpunkte, Items genannt, sichtbar. Man selektiert einen Programmpunkt, indem man die rechte Maustaste einfach über dem gewünschten Item losläßt oder ihn mit der linken Maustaste anklickt.

Jedes Item kann ein Untermenü, bestehend aus den sogenannten »SubItems«, beinhalten.

Anwahl über Tastenkombination

Items, wie auch SubItems, sind außer mit der Maus über Tastenkombinationen, bestehend aus der rechten Amigataste und einer beliebigen Zeichentaste, zu erreichen.

Ein komplettes Menü nennt man MenuStrip. Ein solcher MenuStrip ist einem Window fest zugeordnet. Er kann aus mehreren Hauptmenüs - bei Intuition spricht man von »Menus« - bestehen.

Jedes Hauptmenü (Menu oder Menuheader), dargestellt durch einen Titel in der Screen-Titelleiste, setzt sich aus einer Anzahl Items und SubItems zusammen.

All diese Elemente verwaltet Intuition, wie aus Bild 5 hervorgeht, mit verketteten Strukturen.

Man unterscheidet die für Hauptmenüs zuständige Menu-Struktur und die MenuItem-Struktur, welche Items und SubItems beschreibt.

Beginnen wir mit der Menu-Struktur (Listing 34). Viele Komponenten darin kennen Sie bereits von anderen Intuition-Strukturen, sie haben auch hier eine ähnliche Bedeutung. So ist NextMenu nichts anderes als ein Zeiger auf die Struktur des nächsten gleichartigen Intuition-Elementes, in unserem Falle auf die des nächsten Hauptmenüs. LeftEdge und TopEdge beinhalten die Position der linken oberen Ecke des Titels. Die Variable TopEdge sollten sie zur Zeit immer auf 0 setzen. Width und Height beschreiben die Breite und Höhe des Menütitels. Für Height erlaubt die aktuelle Version von Intuition nur die Höhe der Screen-Titelleiste.

Zur Zeit existieren nur zwei Flags für Menüs: MENUENABLED und MIDRAWN. Ein Hauptmenü ist anwählbar, wenn MENUENABLED gesetzt ist. Ansonsten stellt Intuition den Menütitel in Geisterschrift dar. Das komplette Menü ist dann nicht anwählbar.

Den Zustand dieses Flags können Sie jederzeit mit den Funktionen OffMenu(&win,menunumber) und OnMenu(&win, menunumber) ändern. Als Parameter dienen die Adresse des Windows und die Menünummer (siehe unten). Das Flag MIDRAWN setzt Intuition selbst, wenn sich der Mauszeiger bei gedrückter Menütaste über dem Menütitel befindet. Die Variable »MenuName« der Menu-Struktur ist ein Zeiger auf den Menütitel. »FirstItem« enthält die Adresse der ersten Item-Struktur.

Die restlichen vier Variablen der Menu-Struktur brauchen wir nicht zu beachten, sie werden von Intuition verarbeitet.

Alle Menu-Strukturen sind, wie oben kurz angedeutet, miteinander verkettet. Bei der Initialisierung müssen Sie beachten, daß diese Strukturkette von unten nach oben aufgebaut werden muß, da bei der Vereinbarung einer Struktur die nachfolgende schon bekannt sein muß. Diese Art der statischen Initialisierung haben wir bereits bei den Gadgets kennengelernt. Um Ihnen den Einstieg in die Programmierung der Intuition-Menüs zu erleichtern, haben wir in unserem Headerfile Tool.h die Funktion GetMenu(&men,&win,left,width,flags, &titel) eingebaut. GetMenu() (siehe in Listing 1) initialisiert eine Menu-Struktur und setzt sie an den Anfang einer Menükette. Die Parameter von links nach rechts:

men ist der Zeiger auf eine Menu-Struktur;

win ist der Zeiger auf das Window, für das ein Menü eingerichtet werden soll;

left ist die X-Position des Menütitels;

width legt die Breite des Titels fest;

flags beinhaltet die Menu-Flags und

```
struct Menu
{
    struct Menu *NextMenu;
    SHORT LeftEdge, TopEdge, Width, Height;
    USHORT Flags;
    BYTE *MenuName;
    struct MenuItem *FirstItem;
    SHORT JazzX, JazzY, BeatX, BeatY;
};
```

Listing 34. Kurz und bündig: die Menu-Struktur

```
struct MenuItem
{
    struct MenuItem *NextItem;
    SHORT LeftEdge, TopEdge, Width, Height;
    USHORT Flags;
    LONG MutualExclude;
    APTR ItemFill;
    APTR SelectFill;
    BYTE Command;
    struct MenuItem *SubItem;
    USHORT NextSelect;
};
```

Listing 35. Jeder Menüpunkt wird mit einer MenuItem-Struktur festgelegt

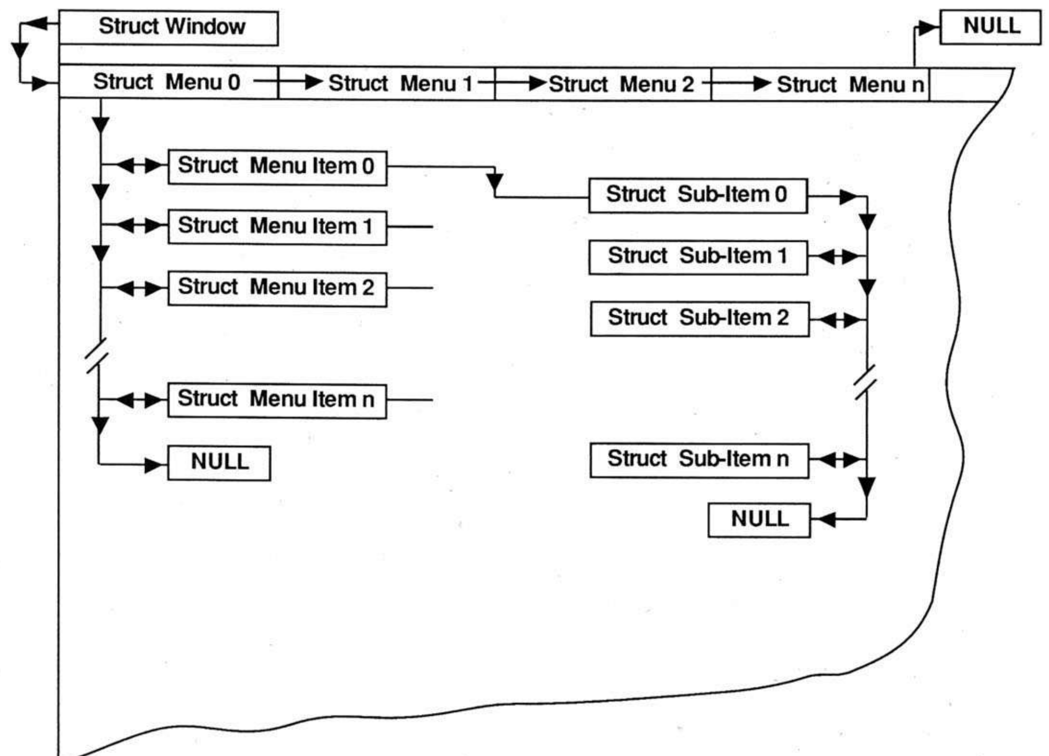


Bild 5. Alle Elemente eines Intuition-Menüs sind verkettet, so daß ein einziger Aufruf ausreicht, um es in einem Window zu installieren

title ist ein Zeiger auf den Menu-Titeltext.

Für die Variable NextMenu übernehmen wir die in der Window-Struktur unter »MenuStrip« gespeicherte Adresse. Dafür sichern wir dort die Adresse der gerade vorliegenden Menu-Struktur. Wir schieben sozusagen die neu initialisierte Struktur an die erste Stelle der Strukturkette. Um die ganze Kette zurückverfolgen zu können, muß ja, wegen der Verkettung, nur die erste Struktur bekannt sein. Aus diesem Grunde dient diese auch als Parameter für die Funktion SetMenuStrip(&win,&menu). Der Aufruf dieser Funktion installiert den angegebenen MenuStrip auf dem Window, dessen Adresse als erster Parameter übergeben wurde. Bei aktivem Window ist nun das Menü ansprechbar. Die Initialisierung eines dreiteiligen Menüs könnte etwa so aussehen:

```
for(i=2;i>=0;i--){
GetMenu(&m[i],&wl,w,f,
t[i]);
}
SetMenuStrip(&w,&m[0]);
```

Wenn wir so vorgehen, liefert, wie wir später noch sehen werden, das Macro MENUNUM() bei der Menüabfrage einen Wert, der dem Index des gewählten Menüs entspricht. Einen Beispielaufruf von GetMenu() finden sie in Listing 36.

Was uns jetzt noch fehlt, sind die Menüpunkte, die sogenannten Items, die bei der Anwahl des Menütitels »aufklappen«.

Funktion und Aussehen dieser Items beschreiben wir, wie schon angedeutet, mit der MenuItem-Struktur (Listing 35).

Alle Items eines Menüs sind über das erste Element der MenuItem-Struktur, dem Zeiger NextItem, miteinander verkettet. Bild 5 zeigt, wie die Variable FirstItem der Menu-Struktur auf das erste Glied der MenuItem-Strukturkette verweist.

Flexible Menüs mit Flags

Die Variablen LeftEdge, TopEdge, Width und Height beschreiben hier die Position und Dimension der Item-Box. Diese Angaben beziehen sich auf die Position des Menütitels in der Titelleiste. Geben Sie also beispielsweise 0 für LeftEdge an, so befindet sich die linke Seite des Items genau unter der linken Seite des Menütitels.

Für die MenuItems stehen uns eine Menge Flags zur Verfügung. Wir besprechen Sie in der im Headerfile intuition.h definierten Reihenfolge:

»CHECKIT« Das Item wird zu einem Ein-/Aus- Schalter, einem sogenannten Attribut-Item. Der aktuelle Zustand wird mit einem Häkchen angezeigt.

»ITEMTEXT« In der Itembox soll Text dargestellt werden. Der Zeiger ItemFill (siehe unten) zeigt in diesem Fall auf eine IntuiText-Struktur.

»COMMSEQ« Setzen Sie dieses Flag, wenn das Item auch mit einer Tastenkombination anwählbar sein soll. Die Kombination besteht aus der rechten Amigataste und einer beliebigen Zeichentaste. Kombinationen dieser Zeichentaste mit der <ALT>-Taste sind erlaubt. Zwischen geshifteten und ungeshifteten Tasten wird nicht unterschieden. Geben Sie das gewünschte Zeichen unter »Command« in der MenuItem-Struktur an. Intuition zeigt dann die Tastenkombination in der Item-Box an.

»MENUTOGGLE« Ist dieses Flag gesetzt, kehrt Intuition bei der Anwahl eines Attribut-Item den Zustand des CHECKED-Flags um und zeichnet (oder löscht) das Häkchen entsprechend.

»ITEMENABLED« Man kann das Item nur anwählen, wenn ITEMENABLED gesetzt ist (siehe auf MENUENABLED). Wie Menüs können auch Items mit der Funktion OffMenu() aus- und mit OnMenu() eingeschaltet werden. Hierzu erfahren Sie später mehr.

»HIGHFLAGS« Hat den gleichen Effekt wie HIGHNONE. Man sollte HIGHFLAGS nicht setzen, denn es handelt sich nur um einen Überbegriff für die vier folgenden Flags. Diese beschreiben das Aussehen der Items bei der Anwahl:

»HIGHIMAGE« Das unter SelectFill (siehe unten) beschriebene Element wird angezeigt.

»HIGHCOMP« Anzeige des Items in Komplementärfarben.

»HIGHBOX« Zeichnet einen Rahmen um die Itembox.

»HIGHNONE« Bewirkt keine Veränderung bei der Anwahl.

»CHECKED« Ist dieses Flag gesetzt, zeichnet Intuition vor den Itemtitel eines Attribut-Items ein Häkchen. Lassen Sie daher vor dem Titel, etwa durch das Voranstellen zweier Leerzeichen, ausreichend Platz. Möchten Sie gerne ein eigenes Häkchen verwenden, initialisieren Sie den Zeiger »CheckMark« der Window-Struktur mit der Adresse des entsprechenden Images.

```
/*
\\        CLIcol.c - Demo Pulldown-Menues
*/
#include "TOOL.h"

LONG ex[]={             /* MutualExclude-Werte für die Items der
Menues 2 und 3 */
    14,13,11,7
};
BYTE c[2][4]={                  /* Command-Keys für die Items
der Menues 2 und 3 */
    'B','W','S','R',
    '1','2','3','4'
};
BYTE *Mtitel[]={                            /* Die Titel der Menues 1-3 */
    "CLICol",
    "Vordergrund",
    "Hintergrund"
};
struct IntuiText SI[2]={        /* Text für die SubItems des Item 1
in Menu 1 */
    2,0,JAM1,30,5,NULL,(UBYTE *)"* Gölzy's CLICol *",NULL,
    3,0,JAM1,30,5,NULL,(UBYTE *)"Wirklich beenden ?",NULL
};
struct IntuiText m1[2]={                        /* Text für die Items
in Menu 1 */
    0,0,JAM1,0,0,NULL,(UBYTE *)"About",NULL,
    3,0,JAM1,0,0,NULL,(UBYTE *)"Ende ",NULL
};
struct IntuiText m2[4]={              /* Text für die Items der
Menues 2 und 3 */
    2,0,JAM1,0,0,NULL,(UBYTE *)"  Blau   ",NULL,
    2,0,JAM1,0,0,NULL,(UBYTE *)"  Weiß   ",NULL,
    2,0,JAM1,0,0,NULL,(UBYTE *)"  Schwarz",NULL,
    2,0,JAM1,0,0,NULL,(UBYTE *)"  Rot    ",NULL
};
VOID main()
{
    struct Window *win;
    struct Menu men[3];
    struct MenuItem I1[2],I2[2][4],S1[2];
    SHORT i, ende=0;
    USHORT code,Cfl=ITEMTEXT|ITEMENABLED|HIGHBOX,/* Item-Flags
    in Menu 1 und */
          Ffl=CHECKIT|ITEMTEXT|ITEMENABLED|HIGHCOMP|COMMSEQ; /*
          Menu  2 + 3 */
    ULONG class;
if(OpenLib()) exit(0);
ModifyIDCMP((win=ACTIVE_WINDOW),MENUPICK);
for(i=2;i>=0;i--){ /* TOOL.h Funktion GetMenu(): Menuheader
installieren */
    GetMenu(&men[i],win,i*100,100,MENUENABLED,Mtitel[i]);
}
for(i=1;i>=0;i--){/* MenuItems und SubItems für linkes Menu
installieren */
    GetItem(&men[0],&I1[i],0,i*10,100,10,Cfl,0,&m1[i],NULL,0);
    GetSItem(&I1[i],&S1[i],100,10,200,20,Cfl,0,&SI[i],NULL,0);
}
for(i=3;i>=0;i--){            /* MenuItems für mittleres Menu
installieren */
    GetItem(&men[1],&I2[0][i],0,i*10,120,10,Ffl,ex[i],&m2[i],
    NULL,c[0][i]);
}
for(i=3;i>=0;i--){              /* MenuItems für rechtes Menu
installieren */
    GetItem(&men[2],&I2[1][i],0,i*10,120,10,Ffl,ex[i],&m2[i],
    NULL,c[1][i]);
}
SetMenuStrip(win,&men[0]);/* kompletter MenuStrip in CLI-Window
aufbauen */
while(ende==0){
    Wait(1<<win->UserPort->mp_SigBit);
    class=GetMessage(win,&code,0,0);
    switch(class){
        case MENUPICK:{                         /* Ein Menü wurde
        angewählt */
            if(code!=MENUNULL){
                switch(MENUNUM(code)){          /* Menu-Nummer
                ermitteln */
                    case 0:{ /* 1. Menu in Strukturkette wurde
                    angewählt */
```

Listing 36. CLICol demonstriert: die Tool.h-Funktionen GetMenu(), GetItem() und GetSItem() erleichtert das Arbeiten mit Intuition-Menüs

```
                    switch(ITEMNUM(code)){  /* Item-Nummer
                    ermitteln */
                        case 0: break;      /* Item 1 der
                        Itemkette */
                        case 1:{            /* Item 2 der
                        Itemkette */
                            if(SUBNUM(code)==0) ende ++;
                            /*SubItem-Nr.*/
                            break;
                        }
                    }
                    break;
                }
                case 1:{ /* 2. Menu in Strukturkette wurde
                angewählt */
                    switch(ITEMNUM(code)){
                        case 0:{ /* Item 1 gewählt:
                        Vordergrund Blau */
                            printf("\033[30m%c\n",12);
                            break;
                        }
                        case 1:{ /* Item 2 gewählt:
                        Vordergrund Weiß */
                            printf("\033[31m%c\n",12);
                            break;
                        }
                        case 2:{       /* Item 3:
                        Vordergrund Schwarz */
                            printf("\033[32m%c\n",12);
                            break;
                        }
                        case 3:{   /* Item 4 bedeutet
Vordergrund Rot */
                            printf("\033[33m%c\n",12);
                            break;
                        }
                    }
                    break;
                }
                case 2:{ /* 3. Menu in Strukturkette wurde
                angewählt */
                    switch(ITEMNUM(code)){
                        case 0:{ /* Item 1 gewählt:
                        Hintergrund Blau */
                            printf("\033[40m%c\n",12);
                            break;
                        }
                        case 1:{ /* Item 2 gewählt:
                        Hintergrund Weiß */
                            printf("\033[41m%c\n",12);
                            break;
                        }
                        case 2:{       /* Item 3:
                        Hintergrund Schwarz */
                            printf("\033[42m%c\n",12);
                            break;
                        }
                        case 3:{  /* Item 4 bedeutet
                        Hintergrund Rot */
                            printf("\033[43m%c\n",12);
                            break;
                        }
                    }
                    break;
                }
            } /* switch Menu */
        } /* if MENUNULL */
        break;
      } /* case MENUPICK */
    } /* switch class */
  } /* while */
  ClearMenuStrip(win);                /* MenuStrip aus CLI-Window
  löschen */
  ModifyIDCMP(win,0);  /* IDCMPFlags des CLI-Window wieder auf
  null setzen */
}
```

Listing 36. (Schluß)

Die restlichen drei Flags setzt und löscht Intuition selbst, man kann sie abfragen, um bestimmte Informationen zu erhalten:

»ISDRAWN« Das Item ist sichtbar;

»HIGHITEM« Der Mauszeiger befindet sich über einem Item;

»MENUTOGGLED« Ein Attribut-Image wurde angewählt. Dieses Flag abzufragen ist nur bei gesetztem MENUTOGGLE-Flag sinnvoll.

Mittels eines 32 Bit breiten Wertes – »Mutual-Exclude« – beschreiben wir, welche Attribut-Items eines Menüs sich gegenseitig ausschließen. Was bedeutet das? Angenommen Sie arbeiten mit einem Zeichenprogramm.

In einem Menü stehen dort mehrere Pinselbreiten zur Auswahl. Natürlich kann man immer nur mit einer Pinselbreite zeichnen. Das entsprechende Item ist im Menü mit einem Häkchen gekennzeichnet, das Flag CHECKED ist gesetzt.

Bei der Wahl einer anderen Pinselbreite wird das CHECKED-Flag und damit auch das Häkchen des gerade aktiven Items gelöscht. Intuition setzt das Flag in der Struktur des neu gewählten Items und zeichnet vor den Itemnamen das Häkchen. Die neue Pinselbreite ist aktiv.

In unserem Beispiel schließt jeder Menüpunkt alle anderen aus. Ein Menü setzt sich aus maximal 32 Items zusammen. Jedes Bit des unter MutualExclude gespeicherten Wertes repräsentiert ein Item des Menüs. Ein gesetztes Bit bedeutet den Ausschluß des korrespondierenden Items.

Mehrfachauswahl möglich

Dem ersten Item, die Menu-Strukturkomponente FirstItem verweist darauf, ist Bit 0 zugeordnet, dem zweitem Item Bit 1 etc.

In CLICol (Listing 36) finden Sie eine Anwendung von MutualExclude. Dort schließen sich alle Items in jedem Farbmenü gegenseitig aus.

Die nächste Komponente der MenuItem-Struktur ist der Zeiger »ItemFill«. Er zeigt auf eine Image-Struktur oder, falls das ITEMTEXT-Flag gesetzt ist, auf eine IntuiText-Struktur. Image oder Text sind sichtbar, solange sich der Mauzeiger nicht über dem Item befindet.

»SelectFill« verweist dagegen auf die Struktur eines Elementes (Text oder Image), das bei der Anwahl des Items sichtbar sein soll. Voraussetzung dafür ist allerdings, daß das HIGHIMAGE-Flag gesetzt ist.

»Command« beinhaltet das Zeichen der Taste, die zusammen mit der rechtem Amigataste ebenfalls zur Anwahl eines Items führt. Die Variable SubItem beinhaltet die Adresse eines Untermenüpunktes (SubItem) oder NULL, wenn das Item über keine Untermenüs verfügt.

Die letzte Variable der MenuItem-Struktur ist »NextSelect«. Intuition legt dort bei einer Mehrfachauswahl, also der Anwahl mehrerer Menüpunkte mit der linken Maustaste, die sogenannte »MenuNumber« des nächsten Item oder SubItem ab. Diese Menünummer ist ein 16-Bit-Wert. Die fünf oberen Bits stehen für das gewählte SubItem, die nächsten sechs für das Item und die fünf niederwertigen Bits für das aus dem MenuStrip gewählte Menu.

Hat der Anwender einen oder mehrere Menüpunkte gewählt, sendet uns Intuition über den Messageport eine Nachricht von der Class »MENUPICK«. Die Variable Code der Message-Struktur enthält nun die Menünummer der letzten Wahl. Falls aber mehrere Menüpunkte selektiert wurden, finden wir unter NextSelect in der MenuItem-Struktur des gerade zu bearbeitenden Items (SubItems) eine weitere Menünummer:

```
menunumber=item->
NextSelect;
```

Diese MenuNumber steht für die vorletzte Wahl. Doch wie kommen wir zu der Adresse des betreffenden Items? Intuition hält hierfür eine spezielle Routine bereit: ItemAddress(&menu, menunumber). Der erste Parameter ist ein Zeiger auf das erste Menü im MenuStrip, wie bei der Funktion SetMenuStrip() angegeben. Weiter müssen wir noch den Wert der Message-Strukturvariablen Code, also die MenuNumber, übergeben. Wenn die Möglichkeit der Mehrfachauswahl erhalten bleiben soll, muß das Programm nacheinander alle Strukturen solange prüfen, bis es als Wert für NextSelect das Symbol MENUNULL findet. Folgendes Programmfragment übernimmt diese Aufgabe:

```
menunumber=Message->
Code;
while(menunumber !=
MENUNULL){
  item=ItemAddress
  (&menu,menunumber);
  menunumber=item->
  NextSelect;
}
```

Auch die MenuItem-Strukturen muß man bei der Initialisierung von unten nach oben aufbauen. Einfacher und platzsparender, – vor allem dann, wenn man mit vielen MenuItems arbeitet, – ist jedoch die Methode, alle Strukturen in einer speziel-

len Funktion mit Werten zu belegen. Sie haben es sich sicher schon gedacht – solch eine Funktion finden Sie in Tool.h:

GetItem(&m,&i,l,t,w,h,f,me,&If,&Sf,c) arbeitet im Prinzip wie GetMenu(). Wir benutzen hier die Variable FirstItem der Menu-Struktur als Zwischenspeicher zum Aufbau der Item-Kette. Auch hier gilt: die zuletzt initialisierte Struktur ist das erste Glied der Struktur-Kette.

Einfacher Aufruf mit Get-Item

Die Funktion verlangt folgende Parameter:

&m ist die Adresse des Menüs, unter dem das Item erscheinen soll;

&i übergibt die Adresse der MenuItem-Struktur;

l,t,w,h legen Position und Dimension des Items fest;

f sind die Flags;

me nimmt die Flags für MutualExclude auf;

&If und **&Sf** sind die Zeiger ItemFill und SelectFill;

c ist der Shortcut für das Commandzeichen.

Listing 38 demonstriert den Aufruf der neuen Tool.h-Funktion GetItem(). Bevor wir uns näher mit dem Beispielprogramm beschäftigen, besprechen wir noch das letzte Menü-Element – das SubItem.

Durch die Verwendung von SubItems werden Items zu Untermenütiteln. Bewegt man den Mauszeiger über einen Untermenütitel, öffnet sich ein Untermenü welches wieder aus mehreren Auswahlmöglichkeiten, den sogenannten SubItems, bestehen kann.

SubItems werden mit der uns bekannten MenuItem-Struktur beschrieben. Alle SubItems sind wieder mit der jeweils ersten Strukturkomponente miteinander verkettet. Die MenuItem-Strukturvariable SubItem verweist auf das erste Glied dieser Strukturkette.

Die Handhabung der SubItems ist völlig identisch mit der der MenuItems. Eine weitere Verschachtelung in SubSubItems ist zwar denkbar. Intuition ignoriert jedoch die SubItem-Variable eines SubItems. GetSItem() hilft Ihnen bei der Installierung eines Untermenüs. Die Funktion GetSItem(&i,&s,l,t,w,h,f,m,&If,&Sf,c), die Sie ebenfalls in Tool.h finden, verlangt fast die gleichen Parameter wie GetItem().

Die Unterschiede:

&i ist der Zeiger auf das Item, unter dem sich das SubItem einordnen soll;

&s verweist auf das SubItem selbst. Die restlichen Parameter haben die gleiche Bedeutung wie weiter oben für die Funktion GetItem() erläutert.

Hat man alle Menu- und MenuItem-Strukturen vereinbart, installiert ein Aufruf der Funktion SetMenuStrip() das Menü in einem Window. Das Gegenstück dazu, ClearMenuStrip (&win), löscht das Menü. Die Funktion fordert lediglich den Zeiger auf das Window als Argument.

Wie weiter oben schon angesprochen, benachrichtigt uns Intuition über den Messageport, wenn ein Menü bedient wurde. Hat der Anwender die rechte Maustaste betätigt, aber kein Element aus dem Menü selektiert, so finden wir in der IntuiMessage-Strukturvariablen Code das Symbol MENUNULL.

Ansonsten hinterlegt Intuition dort die Menünummer, aus welcher sich der gewählte Menüpunkt ermitteln läßt. Im Headerfile intuition.h sind für diesen Zweck drei Macros definiert:

```
MNr=MENUNUM(Code);
INr=ITEMNUM(Code);
SNr=SUBNUM(Code);
```

Jedes Macro liefert einen Wert von 0 bis n, der der Nummer der Struktur innerhalb der jeweiligen Strukturkette entspricht. Wurde kein SubItem registriert, erhalten wir den Wert »NOSUB«. Dementsprechend kehren die Macros mit »NOMENU« oder »NOITEM« zurück, wenn sie keine Menü- beziehungsweise Itemnummer ermitteln konnten.

Auch der umgekehrte Weg ist möglich:

```
MCode=SHIFTMENU(Nr);
ICode=SHIFTITEM(Nr);
SCode=SHIFTSUB(Nr);
```

A und O: die Menünummer

Diese Macros errechnen aus einer bekannten Menu-, Item- oder SubItemnummer den entsprechenden Code und sind daher im Zusammenhang mit den Funktionen OffMenu() und OnMenu() von besonderer Bedeutung. Wie Sie wissen, fordern beide Funktionen neben dem Windowpointer die Menü-Nummer als Argument.

Durch Addition der verschiedenen Codes läßt sich die Menünummer leicht errechnen. Möchte man beispielsweise das Item 1 im Menü 1 ausschalten, könnte der Funktionsaufruf etwa so aussehen:

```
menunumber=SHIFTMENU(1)+
SHIFTITEM(1);
OffMenu(&win,
menunumber);
```

Der Wert, der als Parameter in OffMenu() das komplette Menu mit der Nummer 1 ausschaltet, errechnet sich aus: SHIFTMENU(1)+SHIFTITEM (NOITEM).

Schauen wir uns das alles an einem praktischen Beispiel an. CLICol, unser Listing 38, öffnet im aktiven CLI-Fenster einen MenuStrip, der die Wahl verschiedener Farben für Vorder- und Hintergrund erlaubt. Zu Beginn des Programms legen wir in verschiedenen Arrays die Menü-Titel und die IntuiText-Strukturen der Items an. In der Hauptfunktion main() vereinbaren wir dann die notwendigen Menu- und MenuItem-Strukturen. Nach dem Öffnen der Libraries setzt die Funktion ModifyIDCMP() im CLI-Fenster das MENUPICK-Flag. Jetzt steht der Installation eines MenuStrips im CLI-Fenster nichts mehr im Wege. In einer Schleife legt uns GetMenu() drei Menüs, men[2] bis men[0], an. Menü men[0] ist das zuletzt definierte und dient daher später als Parameter beim Aufruf der Funktion SetMenuStrip(). Wird men[0] gewählt, liefert das Macro MENUNUM() den Wert 0, dem Index entsprechend.

Es folgt die Initialisierung der Items für men[0] und deren SubItems. Die beiden nächsten Schleifen bauen die Items der restlichen Menüs auf. Die Flags, in der Variablen »Ffl« gespeichert, geben darüber Auskunft, daß es sich um Attribut-Items mit Text und Kommandosequenzen handelt.

Die MutualExclude-Werte, im Array ex[] zu finden, verraten, daß sich die Items gegenseitig ausschließen. Wenn Sie mit MutualExclude noch Probleme haben, schreiben Sie sich am besten die Werte doch einmal in binärer Form untereinander – dann fällt Ihnen sicher die Korrespondenz zwischen gesetzten Bits und Menüpunkten auf.

Der Funktion SetMenuStrip() schließt sich eine while-Schleife an, in deren Rumpf das Programm auf eine Intuition-Nachricht wartet. Ist diese Nachricht gleich MENUPICK, so ermitteln wir über MENUNUM() das gewählte Menü.

Für jedes Menü ist innerhalb der switch-Anweisung ein case-Zweig eingerichtet. In jedem Zweig prüfen wir, wiederum mit einer switch-Anweisung, welches Item registriert wurde. Als Argument dient uns dabei das Resultat des Macros ITEMNUM().

Entsprechend dem gewählten Item schalten die sogenannten »Console-Control-Sequenzen« Vorder- oder Hintergrundfarbe um. Im Menüpunkt Ende des ersten Menüs ist eine »Sicherheitsabfrage« in Form eines SubItems eingebaut. Wählt man das SubItem, so erfüllt die Inkrementierung der Variablen »ende« das Schleifenkriterium. Es folgt, nachdem MenuStrip und IDCMP-Flags gelöscht sind, das Programmende.

Versuchen Sie doch einmal das Programm zu erweitern oder zu verändern. Da es ohnehin für das CLI-Fenster konzipiert ist, könnte man in Verbindung mit der Execute()-Funktion häufig benutzte CLI-Kommandos oder sogar Kommandofolgen, wie zum Beispiel Compiler-Optionen, einbauen. Wenn Sie CLICol mit »RUN« starten, können Sie problemlos im CLI weiterarbeiten.

Dem Guru auf den Zahn gefühlt

Nachdem nun die Menütechnik besprochen ist, könnten Sie auch Ihre eigenen Programme, etwa ein Malprogramm, mit Intuition-Menüs bereichern.

Es gibt sicher noch einiges zu tun, wenn wir alle Systemroutinen in den Griff bekommen wollen. Aber je mehr Sie mit den Systemroutinen arbeiten, um so weniger Fehler machen Sie.

Apropos Fehler – hatten Sie auch schon mal beim Start eines eigenen Programms solch ein lustiges rotes Kästchen, das auf schwarzem Grund hämisch blinkend den Systemabsturz ankündigt?

Der Aufforderung »Press left mouse button to continue« kommt man in der Aufregung oft zu schnell nach, die angegebene Gurunummer verschwindet unbeachtet. Dabei kann man mit Hilfe dieser Nummer den aufgetretenen Fehler einkreisen, oft sogar genau bestimmen.

Leider ist in keinem der mitgelieferten Handbücher eine detaillierte Beschreibung der möglichen Ziffernkombinationen zu finden. Grund genug, einmal im Rahmen eines Programmierkurses darauf einzugehen. Schließlich werden Sie bei kaum einem anderen Computer so komfortabel bei der Fehlersuche unterstützt.

Es handelt sich bei der Gurunummer nicht etwa um eine riesige Fließpunktzahl, sondern

Fehlerklassen der Gurus

BB	Fehlerklasse
01	kein Speicher
02	Library konnte nicht erzeugt werden
03	Library läßt sich nicht öffnen
04	Device läßt sich nicht öffnen
05	Hardware-Fehler
06	I/O-Fehler
07	kein I/O-Signal

Tabelle 14. Erste Hinweise auf den Fehler: die Fehlerklasse der Gurunummer

um eine Kombination von Ziffern, die zusammen Hinweise auf die Fehlerursache und den Fehlerort geben können.

Die Nummer hinter dem vermeintlichen Dezimalpunkt stellt die Adresse des fehlerhaften Tasks in hexadezimaler Schreibweise dar. Der Task, der wegen eines Fehlers die folgende Nummer in einer Alarmmeldung erscheinen läßt,

```
84010002.00203750
```

beginnt ab der Adresse $203750. Die acht Ziffern vor dem Punkt deuten auf den Fehler selbst hin. Man kann sie in drei Gruppen einteilen:

```
AABBCCCC
```

Das erste Byte, hier bezeichnet mit »AA«, teilt uns mit, in welchem Teil des Betriebssystems der Fehler auftrat. Byte »BB« enthält die Fehlerklasse in Form einer allgemeinen Diagnose (siehe Tabelle 14). Speicherwort »CCCC«, die vier Ziffern vor dem Punkt, beschreibt spezielle Fehler des unter »AA« aufgeführten SubSystems.

Wenn möglich, gibt die erste Ziffer (Nibble) von »AA« die Art der Alarmmeldung, auch Alert genannt, an. Wir unterscheiden 0 für rückkehrfähige (RECOVERABLE_ALERT) und 8 für nicht rückkehrfähige Alerts (DEADEND_ALERT).

Die erste Gruppe ist oft, neben der führenden 0 der Gurunummer, daran zu erkennen, daß sie den aktuellen Screen nur um die Höhe des Alerts nach unten schiebt. Nach einem Klick auf die rechte Maustaste erfolgt der Rücksprung ins Programm. Meist kann man dann weiterarbeiten.

Deadend Alerts kündigen den Systemabsturz auf schwarzem Bildschirm an. Hier wird nach Betätigung der rechten Maustaste der interne Debugger aufgerufen, welchen man mit einem am seriellen Port angeschlossenen Terminal bedienen kann.

Folgt man jedoch der freundlichen Aufforderung, die linke Maustaste zu betätigen, so erfolgt in beiden Fällen der Neustart des Systems. Betrachten wir nun Byte »AA« genauer. In Tabelle 15 sind alle möglichen (genauer: uns bekannten) Codes den entsprechenden Betriebssystemteilen gegenübergestellt.

Alle Ziffern mit einer führenden Null stehen dabei für Libraries, die mit einer Eins beginnen für Devices und solche mit einer zwei am Anfang für Resources.

Hinter dem SubSystem sind, soweit vorhanden (beziehungsweise uns bekannt) die jeweiligen speziellen Fehler aufgelistet. Sie sind, wie erwähnt, mit dem Speicherwort »CCCC« vor dem Punkt der Gurunummer beschrieben.

Der Guru als Debugginghilfe

Sind die ersten beiden Ziffern, also »AA«, gleich Null, so handelt es sich um eine Prozessor-Fehlermeldung (CPU- oder Prozessor-Trap). In diesem Fall ist auch die Fehlerklasse gleich 0. Der Fehler wird mit einer einzigen Ziffer, der Trap-Nummer, beschrieben. In Tabelle 16 sind die möglichen Werte aufgelistet.

Wie Sie der Tabelle entnehmen können, existieren Traps, bei denen es sinnvoll ist, sie absichtlich zu erzeugen. Ein Beispiel dafür ist der Trace-Modus.

Vielleicht fotokopieren Sie sich die Tabellen und hängen sie über dem Monitor auf. So sind Sie in der Lage, alle Gurunummern zu entschlüsseln und können sich Zeit und Ärger sparen. Hier noch zwei Beispiele:

```
04010003.00203750
```

Der Fehler trat – die 4 verrät es – in der Intuition-Library auf. Der Grund war (01) Speichermangel, genauer (0003) fehlender Speicher zur Darstellung eines Untermenüs. Der fehlerhafte Task beginnt ab Adresse $203750.

```
82010008.00203750
```

Die 8 am Anfang der Nummer kündigt den bevorstehenden Systemabsturz an. Der Fehler wurde in der Graphics-Library (2) lokalisiert, weil kein Speicherplatz (01) zum Füllen (0008) zur Verfügung stand. Solche Fehler können beispielsweise dann auftreten, wenn vor der Anwendung der Flood()-Funktion kein oder ein zu kleines Temporärraster vereinbart wurde.

Mit den oben besprochenen Fehlermeldungen in Form eines Alerts, quittiert uns das Betriebssystemteil Exec unsere Fehler. Demnach müssen dort auch die entsprechenden Funktionen zu finden sein, um solche Alerts zu erzeugen.

Tatsächlich stellt uns Exec eine Funktion namens Alert(Nr,n) zur Verfügung, mit deren Hilfe wir sehr einfach die rotblinkenden Alarmmeldungen in eigene Programme einbinden können.

Als ersten Parameter verlangt sie die Nummer, die als Guru-Nummer im Alert erscheinen soll. Im Headerfile exec/alerts.h sind verschiedene Nummern als symbolische Konstanten vereinbart. Wir unterscheiden vier Nummerntypen:

AT-Nummern geben an, ob das Alert rückkehrfähig ist oder nicht, AO-Nummern bezeichnen den Fehlerbereich, AG-Nummern liefern die Fehlerklasse und AN-Nummern den genauen Fehler, auf den Fehlerbereich bezogen.

AN-Nummern eignen sich direkt als Parameter für Alert(), jedoch kann man sich durch »ODER«-Verknüpfung anderer Symbole eigene Nummern zusammenstellen.

Exec schiebt den aktiven Screen um die Höhe des Alerts nach unten, wenn der Parameter n der Alert()-Funktion gleich Null ist. Ein Neustart erfolgt nur wenn der erste Parameter das Symbol AT-DeadEnd beinhaltet.

Gibt man für n eine eins oder eine andere ungerade Zahl an, erscheint das bekannte »Software error«-Requester. Der Alert präsentiert sich nach der Anwahl von »Cancel«, auf schwarzem Bildschirm. Danach erfolgt der Neustart des Systems. Hier ein vergleichsweise »harmloser« Beispielaufruf der Funktion:

```
#include <exec/
alerts.h>
main()
{
 Alert(AN_TextTmpRas,0);
}
```

Nicht sehr komfortabel, aber wirkungsvoll und schnell in eigene Programme installiert. Intuition-Alerts bieten allerdings schon etwas mehr. Die Funktion DisplayAlert(Art,&Text, height) schiebt den aktuellen Screen soweit nach unten, daß eine genau »height« Pixel hohe Alarmmeldung Platz findet.

Die Alertart gibt an, ob es sich um einen rückkehrfähigen (RECOVERY_ALERT) oder nicht rückkehrfähigen (DEADEND_ALERT) Alert handeln soll. Setzen Sie eines der in Klammern angegebenen Symbole ein. Parameter zwei ist ein Zeiger auf eine Zeichenkette, die den Alert-Text beinhaltet. Innerhalb dieser Zeichenkette erwartet Intuition aber noch andere Informationen (siehe auch Listing 39):

Die drei ersten Bytes beschreiben die X- und die Y-Posi-

Die Prozessor-Traps

AABBCCCC	CPU-Trap
00000002	Busfehler; extern erzeugtes Eingangssignal zeigt Hardwarefehler an
00000003	Adressierungsfehler; Word-Operand an ungerader Adresse
00000004	Illegale Anweisung; kein gültiger 68000-Befehl
00000005	Division durch Null; verbotene Division mit Divisor gleich Null
00000006	CHK-Befehl; Wert eines Datenregisters nicht im Bereich
00000007	TRAPV-Befehl; gesetztes Überlaufbit
00000008	Nutzung der Privileg-Befehle außerhalb des Supervisor-Modus
00000009	Trace-Modus; schrittweise Abarbeitung bei gesetztem Trace-Bit
0000000A	Line Emulator; OPCode beginnt mit der Bitfolge 1010
0000000B	Line Emulator; OPCode beginnt mit der Bitfolge 1111

Tabelle 16. Diese Fehlermeldungen löst der Prozessor selbst aus. Sie sind nie vom Typ »Recoverable«

Die Gurus entschlüsselt

AA	SubSystem	CCCC	Fehler
01	Exec	0000	Checksum Prozessor
		0002	Checksum Exec-Startadresse
		0003	Checksum Library
		0004	kein Speicher für Library
		0005	Fehler in Speicher-Liste
		0006	kein Speicher für Interrupt
		0007	Zeigerfehler
		0008	fehlerhafte Semaphore
		0009	Versuch mehrfacher Speicherfreigabe
		000A	Zeigerfehler bei Ausnahmebedingung
02	Graphics	0001	kein Speicher für Copper-Liste
		0002	kein Speicher für Copperinstruction-Liste
		0003	Copperliste voll
		0004	fehlerhafte Copperliste
		0005	kein Speicher für Kopf der Copper-Liste
		0006	kein Speicher für »long frame«
		0007	kein Speicher für »short frame«
		0008	kein Speicher für Fill-Rekursion
		0009	kein Speicher für Textausgabe
		000A	kein Speicher für BitMap
		000B	fehlerhafte Angabe für Speicherbereich
		0030	konnte kein ViewPort einrichten
		1234	Bereich für Zwischenspeicher belegt
03	Layers	0001	kein Speicher für Layer
04	Intuition	0000	kein oder unbekannter Gadgettyp
		0001	Gadgetfehler
		0002	kein Speicher für Port
		0003	kein Speicher für Darstellung des Menüs
		0004	kein Speicher für Darstellung Untermenü
		0005	kein Speicher für Menü-Titelzeile
		0006	falsche Position der Titelzeile
		0007	kein Speicher für Screen
		0008	kein Speicher für RastPort
		0009	kein oder unbekannter Screentyp
		000A	kein Speicher für Gadget
		000B	kein Speicher für Window
		000C	falscher Systemstatus
		000D	ungültige IDCMP-Message
		000E	Überlauf des Messagepuffers
		000F	kein Speicher für Console-Device
05	Math		
06	CList		
07	DOS	0001	kein Speicher für Start
		0002	Task nicht beendet
		0003	Packet fehlerhaft
		0004	unerwartete oder fehlerhafte Daten
		0005	Zeigerfehler
		0006	Fehler in Block
		0007	Fehler in BitMap
		0008	Versuch mehrfacher Key-Freigabe
		0009	Checksum-Fehler
		000A	Disk-Fehler
		000B	Key außerhalb des Bereichs
		000C	fehlerhafte Überlagerung
08	RAM	0001	Fehler in Speicher-Liste
09	Icon		
0A	Expansion	0001	Hard- oder Softwarefehler bei Erweiterungen
10	Audio		
11	Console		
12	Gameport		
13	Keyboard		
14	Trackdisk	0001	Fehler beim Suchen auf Diskette
		0002	Timerimpuls fehlerhaft
15	Timer	0001	Zugriffversuch fehlerhaft
		0002	Schwankungen der Netzfrequenz
20	CIA		
21	Disk	0001	fehlerhaftes DiskChange-Signal
		0002	fehlerhaftes oder fehlendes Laufwerk
22	Misc		
30	Bootstrap	0001	Boot-Fehler
31	Workbench		
32	Diskcopy		

Tabelle 15. Die Ziffern unter »AA« geben an, wo der Fehler »CCCC« aufgetreten ist

tion der nachfolgenden Textzeile (siehe auch Kommentarzeile in Listing 37). Dabei interpretiert Intuition das erste Byte als das High-Byte der X-Position, das zweite Byte als deren LowByte und das dritte Byte als die Y-Position.

Die maximalen Ausmaße eines Alerts sind 640x256 Punkte. Daher sind für die X-Position zwei Bytes und für die Y-Position nur ein Byte nötig. Es folgt, – mit einem Nullbyte abgeschlossen – der eigentliche Alert-Text.

Hinter dem Text gibt ein einzelnes Byte an, ob ein weiterer Text folgen soll. Ist dieses Byte ungleich null, so folgt die Definition einer weiteren Textzeile, wieder beginnend mit der Positionsbeschreibung. Diese Sequenz wiederholt sich so lange, bis hinter dem Alert-Text eine weitere Null folgt.

Sie sehen, auch Intuition-Alerts sind einfach in eigene Programme einzubinden. Trotzdem sollten Sie sparsam damit umgehen, um den ohnehin schon Alert-geplagten Amiga-User nicht noch unnötig zu erschrecken.

Wie Alerts zählen auch Requester zu den Systemmeldungen. Sie sind jedoch durch die Möglichkeit der Montage mehrerer Gadgets ungleich komfortabler. Man spricht eher von Dialogfeldern.

Dialogfeld Requester

Im einfachsten Fall nehmen sie, wie etwa die Systemrequester, lediglich eine Ja-/Nein-Entscheidung entgegen. Allerdings sind in Bezug auf Anwendungsgebiet und Leistung nach oben hin keine Grenzen gesetzt.

Beispiele hierfür sind die komfortablen Filerequester oder solche zum Einstellen der Bildschirmfarben in den bekannten Malprogrammen.

Man unterscheidet zwei Requesterarten: »AutoRequester« und frei definierbare (»Custom«-)Requester. Die letzteren könnte man wiederum in normale und in »DoubleMenu«-Requester untergliedern. Grundsätzlich weisen aber beide die gleiche Struktur auf.

Der einzige Unterschied: ein DoubleMenu-Requester erscheint nach einem Doppel-Klick mit der rechten Maustaste. Beginnen wir mit den AutoRequestern. Hierzu sind keinerlei Vorbereitungen nötig, der Aufruf der Funktion AutoRequest(&w,&tt&pt,&nt,pf,nf,b,h) öffnet einen Requester mit zwei

```
/*
\\  IAlert.c - demonstriert DisplayAlert()
*/
#include "TOOL.h"

BYTE AlertText[]={
/*X-hi,lo,  Y,  Text in Hochkomma = mit 0 abgeschlossen,
weiter? (1/0) */
    0,140, 20, "Keine Panik, dies ist ein Recovery-Alert!",  1,
    0,190, 45, "//      -*-  Klick  -*-       //",           1,
    0,175, 52, "\\X/                         \\X/",          0
};

VOID main()
{
    if(!(OpenLib())){
       DisplayAlert(RECOVERY_ALERT,&AlertText[0],70);
       CloseLib();
    }
}
```

Listing 37. Kurz und bündig: Intuition-Alerts mit DisplayAlert()

```
/*
\\  AReq.c - demonstriert den Aufruf eines AutoRequesters
*/
#include "TOOL.h"

struct IntuiText RT[3]={
    2,0,JAM1,15,5,NULL,(UBYTE *)"Bildschirm löschen?",NULL,
    3,0,JAM1, 6,4,NULL,(UBYTE *)" Ja ",NULL,
    2,0,JAM1, 6,4,NULL,(UBYTE *)"Nein",NULL
};

VOID main()
{
    if(!(OpenLib())){
      if(AutoRequest(NULL,&RT[0],&RT[1],&RT[2],VANILLAKEY,0,200,50))
           putchar(12);          /* Bildschirm löschen */
       CloseLib();
    }
}
```

Listing 38. Die Funktion AutoRequest() öffnet einfache Requester, die eine Ja-/Nein-Entscheidung entgegen nehmen können

```
struct Requester
{
    struct Requester *OlderRequest;
    SHORT LeftEdge, TopEdge;
    SHORT Width, Height;
    SHORT RelLeft, RelTop;
    struct Gadget *ReqGadget;
    struct Border *ReqBorder;
    struct IntuiText *ReqText;
    USHORT Flags;
    UBYTE BackFill;
    struct Layer *ReqLayer;
    UBYTE ReqPad1[32];
    struct BitMap *ImageBMap;
    struct Window *RWindow;
    UBYTE ReqPad2[36];
};
```

Listing 39. Komfortablere Requester mit der Requester-Struktur

beschrifteten Gadgets und einem Titeltext. Die Parameter:

w Ist ein Zeiger auf eine Window-Struktur, Intuition legt beim Öffnen des Requesterwindows die angegebene Struktur zugrunde. Entsprechend erscheint der dort angegebene Titel auch als Windowtitel im Requester. Geben Sie NULL als Zeiger an, greift Intuition auf die Struktur der Systemrequester-Windows zurück. Wir erhalten dann »System Request« als Windowtitel (Listing 38).

Die folgenden drei Parameter sind Adressen von IntuiText-Strukturen. Dabei bedeutet:

tt die Requesterüberschrift;

pt der Positiv-Text (im linken Gadget) und

nt der Negativ-Text (im rechten Gadget). Weitere Parameter sind **pf** und **nf** die IDCMP-Flags, die die Wahl des Positiv- oder Negativ-Gadgets ersetzen, sowie **b** und **h** die Breite und die Höhe des Requesters.

Der Rückgabewert der Funktion unterscheidet sich nur dann von Null, wenn Intuition die Wahl des Positiv-Gadgets oder, über den Message-Port, das Positiv-Flag registriert hat.

Sehen wir uns Listing 40 etwas genauer an. Die notwendigen IntuiText-Strukturen für die Texte intialisieren wir in einem Array. In der vereinbarten Reigenfolge dienen sie beim Aufruf als Parameter.

Da wir bei Flags »VANILLAKEY« gesetzt haben, kommt ein beliebiger Tastendruck der Anwahl des Positiv-Gadgets gleich.

unvergleichlich flexibel

Eine positive Entscheidung erfüllt die Bedingung der if-Anweisung – der Rückgabewert von AutoRequest() ist ungleich null – die Funktion putchar() löscht den Bildschirm.

AutoRequest() ruft eine Funktion namens BuildSysRequest(w,&tt,&pt,&nt,f,b,h) auf. Die Parameter sind dabei, bis auf »f«, die gleichen wie schon bei AutoRequest() besprochen.

BuildSysRequest() installiert einen Requester mit den beiden Gadgets, kehrt aber mit einem Windowpointer zurück. Als IDCMP-Flags dieses Windows ist der beim Funktionsaufruf als Parameter »f« übergebene Wert zu finden.

Die Abfrage der Gadgets müssen Sie selbst übernehmen. Die Funktion FreeSysRequest(&win) schließt solche Requester. Als Parameter verlangt sie die von BuildSysRequest() gelieferte Adresse.

Möchte man in seinen Programmen komfortable, fast menüähnliche Requester, mit vielen Auswahlmöglichkeiten anbieten, muß man nur ein klein wenig tiefer ins System hinabsteigen und sich mit der Requester-Struktur (Listing 39) befassen.

Packen wir's an: der Zeiger »OlderRequester« interessiert nur Intuition. Er verweist auf den zuvor geöffneten Requester. LeftEdge, TopEdge, Width und Height stehen für die Position und die Dimension des neuen Requesters.

Wenn das Flag »POINTREL« (siehe unten) gesetzt ist, erscheint ein DoubleMenu-Requester relativ zur Mausposition. »RelLeft« und »RelTop« enthalten die entsprechenden Offsetwerte. »ReqGadget« ist die Adresse des ersten Gadgets der Gadgetliste. »ReqBorder« verweist auf die Borderstruktur des Requesterrahmens. Geben Sie NULL an, wenn Sie auf eine Umrandung verzichten wollen. »ReqText« enthält die Adresse der IntuiText-Struktur mit dem Requestertext.

Auch Requester kann man über Flags beeinflussen:

»POINTREL« Ein Double Menu-Requester erscheint relativ zur Mausposition.

»PREDRAWN« Ist nur bei der Verwendung einer eigenen BitMap (ImageBMap) nötig.

»NOISYREQ« Ist ein Requester aktiv, blockiert Intuition alle Nachrichten die dieses nicht betreffen. Mit gesetztem NOISYREQ-Flag, kann man nach wie vor Tastatur und Maus über den Message-Port kontrollieren.

Die restlichen Flags setzt und löscht Intuition selbst. Hier die wichtigsten:

»REQOFFWINDOW« Ein Requester-Gadget befindet sich zumindest teilweise außerhalb des Windows.

»REQACTIVE« Sendet Intuition bei einem aktiven Requester.

»SYSREQUEST« Es handelt sich um einen System-Requester. Die Variable BackFill der Requester-Struktur enthält die Hintergrundfarbe des Requesters.

Von den letzten fünf Variablen ist für uns, wenn überhaupt, nur noch »ImageBMap« interessant. Es handelt sich um einen Zeiger auf eine eigene BitMap. Falls Sie eine solche verwenden möchten, müssen Sie auch das Flag »PREDRAWN« setzen.

Natürlich haben wir in Tool.h auch eine Funktion, die zur einfachen Erzeugung eines Requesters dient. Wir nutzen darin nur die Komponenten bis einschließlich BackFill.

Die Funktion GetRequest(&r, &w,l,t,w,h,rl,rt,&g,&b,&i,f,b,T) in Listing 1, öffnet einen beliebigen Requester. Die Parameter sind nahezu identisch mit der Requester-Struktur:

r ist die Adresse einer Requester-Struktur;

w die Adresse des Windows dem die Requester-Struktur zugeordnet sein soll;

l,t,w,h,rl,rt die Positions- und Dimensionsangaben LeftEdge, TopEdge, Widht, Height, RelLeft und RelTop;

g ein Zeiger auf das erste Element der Gadgetliste;

b ein Zeiger auf eine Borderstruktur;

i ein Zeiger auf eine IntuiText-Struktur für ReqText;

f sind die Flags und

b steht für BackFill.

Der letzte Parameter – T – steht für den Requester-Typ. Geben Sie »DMREQ« für DoubleMenu-Requester und »REQ« für normale Requester an. Die beiden Symbole sind in ebenfalls in Tool.h definiert.

Wie funktioniert nun GetRequest()? Zu Beginn initialisiert die Funktion InitRequester() die gesamte Struktur mit NULL, da wir ja nicht alle Komponenten mit Werten versorgen. Nach der Zuweisung der Parameter, rufen wir, falls es sich um einen DoubleMenu-Requester handelt, die Funktion SetDMRequest(&w,&r) auf. Zwei Pointer als Parameter verweisen auf die betreffende Window-Struktur und auf die Requester-Struktur. Die Funktion läßt den Requester aber nicht erscheinen, dies geschieht erst nach einem Doppelklick.

An dieser Stelle gleich das Gegenstück zu SetDMRequest():

ClearDMRequest(&win)

löscht den DoubleMenu-Requester aus der Liste des angegebenen Windows (nicht vom Bildschirm), so daß eine Aktivierung per Doppelklick nicht mehr möglich ist.

Beide Funktionen liefert »TRUE«, wenn sie korrekt arbeiten konnten und im anderen Fall »FALSE«.

Durch die Angabe von »REQ« als letzten Parameter der Funktion GetRequest() stellt die Funktion Request(&r,&w), in der letzten Zeile von Listing 3, den gewünschten Requester sofort auf dem Bildschirm dar.

Die Parameter von Request() sind die gleichen wie bei SetDMRequest(), sie sind lediglich in der Reihenfolge vertauscht. Da GetRequest() beide Funktionen selbst aufruft, sind Fehler durch Verwechslung der Parameter ausgeschlossen.

Ein Requester kann auf zwei Arten wieder geschlossen, das heißt vom Bildschirm gelöscht werden: a) über den Aufruf der Funktion EndRequest(&r,&w) mit den Adressen des Requesters und des Windows als Parameter; b) durch die Betätigung eines ENDGADGETs (siehe auch unter Gadgets).

Requester und Gadgets

Unsere Tool.h-Funktion GetGadget() erkennt selbständig, ob es sich um Window- oder um Requester-Gadgets handelt. Die Verkettung der Window-Gadgets erfolgt problemlos über AddGadget(). Requester-Gadgets müssen wir »von Hand« verketten. Geben Sie hierzu anstelle des Window-Zeigers die Adresse der jeweils nächsten Gadget-Struktur an. Der Aufruf eines Requesters mit drei Gadgets könnte demnach wie folgt aussehen:

```
/*
\\  DMReq.c  -  Positionierung des Bildschirms über
DoubleMenu-Requesters
*/
#include "TOOL.h"
#define RHO 160
#define RBR 250
#define GHO 15
#define GBR 50
#define GS (0xFFFF/100)

struct Window *win;
struct Image image;
struct PropInfo PInfo={
    FREEVERT| FREEHORIZ| AUTOKNOB,0,0,GS,GS,0,0,0,0,0,0
};
SHORT Rpos[]={
    0,0,RBR-5,0,RBR-5,RHO-3,0,RHO-3,0,0
};
SHORT Gpos[]={
    0,0,GBR,0,GBR,GHO,0,GHO,0,0
};
struct Border RRand={
    2,1,2,0,JAM1,5,&Rpos[0],NULL
};
struct Border GRand={
   -1,-1,2,0,JAM1,5,&Gpos[0],NULL
};
struct IntuiText RText={
    2,0,JAM1,15,10,NULL,(UBYTE *)"Bitte ViewOffset einstellen!",NULL
};
struct IntuiText SText={
    2,0,JAM1,5,3,NULL,(UBYTE *)"SLEEP",NULL
};
struct IntuiText EText={
    3,0,JAM1,9,3,NULL,(UBYTE *)"QUIT",NULL
};

VOID sleep()
{/* warten auf IntuiMessage der Klasse REQSET */
    ULONG class;

    do{
        Wait(1<<win->UserPort->mp_SigBit);
    }while((class=GetMessage(win,0,0,0))!=REQSET);
}

VOID main()
{
    struct Requester req;
    struct Gadget Pgad, Sgad, Egad;
    struct Preferences prefs;
    ULONG class;
    USHORT code,id,bool_activations=RELVERIFY| GADGIMMEDIATE|
    ENGADGET;
    BOOL ende=0;

    if(OpenLib()) exit(0);
    GetPrefs(&prefs,sizeof(struct Preferences));  /* Preferences-
    Werte lesen */
    PInfo.HorizPot=GS*(prefs.ViewXOffset+50);    /* Start-Schieber-
    stellung X */
    PInfo.VertPot =GS*(prefs.ViewYOffset+25);    /* Start-Schieber-
    stellung Y */
    ModifyIDCMP((win=ACTIVE_WINDOW),GADGETUP| MOUSEMOVE| REQSET);/*
    IDCMPFlags */
    GetGadget(&Pgad,NULL,25,25,200,100,0,FOLLOWMOUSE,
              REQGADGET| PROPGADGET,&image,NULL,NULL,&PInfo,0);
    GetGadget(&Sgad,&Pgad, 25,135,GBR,GHO, GADGHCOMP,
    bool_activations,
              REQGADGET| BOOLGADGET,&GRand,NULL,&SText,NULL,1);
    GetGadget(&Egad,&Sgad,175,135,GBR,GHO,GADGHCOMP,bool_activations,
              REQGADGET| BOOLGADGET,&GRand,NULL,&EText,NULL,2);
    GetRequest(&req,win,20,20,RBR,RHO,0,0,&Egad,&RRand,&RText,
    0,1,DMREQ);
    sleep();
    while(NOT ende){
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage(win,&code,&id,0);
        if(class==MOUSEMOVE){/* Autoknob in Proportional-Gadget
        wurde bewegt */
            prefs.ViewXOffset=PInfo.HorizPot/GS-50;
            prefs.ViewYOffset=PInfo.VertPot/GS-25;
            SetPrefs(&prefs,sizeof(struct Preferences),FALSE);
            while((class=GetMessage(win,&code,&id,0))==MOUSEMOVE);
        }
        if(class==GADGETUP){/* eines der beiden BooleanGadgets wurde
        gewählt */
            switch(id){
                case 1:{                        /* SLEEP-Gadget
                selektiert */
                    sleep();                            /* warten
                    auf REQSET */
                    break;
                }
                case 2:{                        /* QUIT-Gadget
                selektiert */
                    ende=1;                 /* Schleifenkriterium
                    erfüllen */
                    break;
                }
                default: break;
            }
        }
    }
    ClearDMRequest(win);                    /* DoubleMenu-Requester
    streichen */
    ModifyIDCMP(win,0);
    CloseLib();
}
```

Listing 40. DMReq öffnet einen DoubleMenu-Requester, in dem man Werte der Preferences-Struktur verändern kann

```
GetGadget(&Gad3,
NULL, ...
GetGadget(&Gad2,
&Gad3, ...
GetGadget(&Gad1,
&Gad2, ...
 GetRequest(... &
Gad1, ...
```

Nur die wesentlichen Parameter sind aufgeführt. Die Gadget-Liste ist von unten nach oben aufgebaut, da bei der Definition beispielsweise des Gadgets Gad2, das nachfolgende Gadget, also Gad3, schon bekannt sein muß.

Der Parameter in GetRequest() verweist auf das zuletzt definierte Gadget. Dies ist das erste Glied der Strukturkette. Die Kette endet bei Gad3, mit dem Verweis NULL als Folgegadget. Ein Beispiel für die Definition von Requester-Gadgets finden Sie in Listing 40, welches wir nun etwas genauer unter die Lupe nehmen wollen.

DMReq startet einen Double-Menu-Requester. Nach dem Aufruf des fertig übersetzten Programms (RUN DMReq) geschieht anfangs nichts. Sie erinnern sich, ein DoubleMenu-Requester öffnet sich erst nach einem Doppelklick mit der rechten Maustaste.

Ist dies geschehen, kann man in einem zweidimensionalen Proportional-Gadget die Postion des Bildes auf dem Monitor einstellen. Darüber hinaus stehen zwei weitere Gadgets zur Wahl, mit denen man entscheidet, ob man die Position später nocheinmal verändern möchte (SLEEP) oder nicht (OUIT).

Die Definition von Symbolen und die Initialisierung von Strukturen bilden zusammen fast das erste Drittel des gesamten Quellcodes. Den Rest teilen sich zwei Funktionen.

Die erste davon, sleep(), besteht nur aus einer einzigen Schleife, in der wir den Message-Port überwachen, bis eine Nachricht der Klasse REQSET ansteht. Die andere ist die Hauptfunktion main().

Nach den notwendigen Vereinbarungen und dem Öffnen der Libraries liest die Intuition-Funktion GetPrefs() die Preferences-Struktur (Listing 41) aus. Diese in intuition.h definierte Struktur enthält alle mit dem Workbenchprogramm Preferences eingestellten Werte.

Die Offsetwerte, die man dieser Kopie entnehmen kann, bilden nach einer kleinen Anpassung die Startposition des »Schiebers« im Proportional-Gadget.

ModifyIDCMP() setzt eine Zeile darunter die notwendigen IDCMP-Flags im CLI-Window. Normalerweise sind ja dort keine IDCMP-Flags gesetzt.

Unser Requester soll mit drei Gadgets ausgestattet sein. Die Tool.h Funktion GetGadget() erledigt das für uns. Beachten Sie die Verkettung durch die Angabe des Folgegadgets als zweiten Parameter im Funktionsaufruf.

Die Angabe von »DMREQ« als letzten Parameter von GetRequest() deklariert das neue Dialogfeld als DoubleMenu-Requester. Unser Programm »schläft« in sleep() bis der Anwender den Requester aktiviert. Intuition sendet dann nur noch Nachrichten die den Requester direkt betreffen.

GetMessage() empfängt und beantwortet sie alle, im Rumpf der sich anschließenden while-Schleife.

Registriert Intuition die Betätigung des Proportional-Gadgets, empfangen wir »MOUSEMOVE«. In diesem Fall schreibt SetPreferences() die veränderten Werte zurück. Auch hierzu gleich mehr.

Empfangen wir die Nachricht »GADGETUP«, unterscheidet eine switch-Anweisung zwischen den beiden BOOL-Gadgets.

Je nach Gadget-ID schicken wir das Programm wieder schlafen oder verlassen die Schleife um es zu beenden.

Nur noch wenige Intuition-Funktionen sind zu besprechen, mit den Requestern haben wir das letzte große Intuition-Element beleuchtet. Damit kann man einiges anfangen! Versuchen Sie sich doch einmal an folgendem Problem:

Realisieren Sie einen »Taschenrechner« als Requester. Bauen Sie soviele Funktionen ein, wie ihnen einfallen – die vier Grundrechenarten sollten es zumindest sein. Alles sollte über eine »Tastatur«, sprich Gadgets, zu bedienen sein. Ein String-Gadget dient als Display. Vielleicht simulieren Sie sogar Ihren eigenen Taschenrechner auf dem Amiga?

Bei einer Fehleingabe, zum Beispiel der Division durch null, soll ein Alert den Anwender warnen. Eine Sicherheitsabfrage in Form eines System-Requesters soll erscheinen, sobald der Anwender das Programm verlassen möchte.

Eine große Aufgabe, zugegeben, aber Sie sollten sich dennoch etwas beeilen, denn wir behandeln jetzt gleich die noch fehlenden Intuition-Funktionen.

```
struct Preferences
{
    BYTE FontHeight;            /* Höhe des Zeichensatzes (8 oder 9)
    in Pixel */
    UBYTE PrinterPort; /* Printer an serieller oder paralleler
    Schnittstelle */
    USHORT BaudRate;   /* Übertragungsgeschwindigkeit der RS-232
    (110-19200) */
    struct timeval KeyRptSpeed;             /* Tastenwiederhol-
    geschwindigkeit */
    struct timeval KeyRptDelay;   /* Verzögerung bis zur ersten
    Wiederholung */
    struct timeval DoubleClick;/* Zeit zwischen 2 Klicks eines
    Doppel-Klicks */
    USHORT PointerMatrix[POINTERSIZE];        /* die Grafik des
    Maus-Pointers */
    BYTE XOffset,YOffset;  /* die Position des >>heißen Punktes<
    < im Pointer */
    USHORT color17,color18,color19;               /* Pointerfarben */
    USHORT PointerTicks;               /* die >>übersetzung<< des
    Mauspointers */
    USHORT color0,color1,color2,color3;                /* die
    Workbenchfarben */
    BYTE ViewXOffset,ViewYOffset; /* die Position des Bildes auf
    dem Monitor */
    WORD ViewInitX, ViewInitY;              /* Initialisierungswerte
    des View */
    BOOL EnableCLI;                         /* CLI-Icon ein- oder
    ausgeschaltet */
    USHORT PrinterType;
    UBYTE PrinterFilename[FILENAME_SIZE];    /* der Name des
    Custom-Printers */
    USHORT PrintPitch;              /* die Schriftart, z.B.: Bold,
    Pica, Italic */
    USHORT PrintQuality;     /* die Qualität des Ausdrucks, z.B.:
    NLQ, Draft */
    USHORT PrintSpacing;                           /* Druck-Abstand */
    UWORD PrintLeftMargin,PrintRightMargin;    /* Rand beim Drucken*/
    USHORT PrintImage;              /* normaler oder reverser Ausdruck */
    USHORT PrintAspect;          /* Ausdruck horizontal oder vertikal */
    USHORT PrintShade; /* Art des Ausdrucks: S/W, Grautöne, Farben */
    WORD PrintThreshold;                /* die Graustufe beim Ausdruck */
    USHORT PaperSize;                               /* die Papiergröße */
    UWORD PaperLength; /* die Länge des Papiers in Zeilen gemessen */
    USHORT PaperType;               /* Endlospapier oder Einzelblätter */
    UBYTE SerRWBits;              /* Read (High) oder Write(Low) Bits */
    UBYTE SerStopBuf;             /* Stopbits (H) und Puffergröße (L) */
    UBYTE SerParShk;                     /* Parity (H) und Shake (L) */
    UBYTE LaceWB;                        /* Flimmer-Workbench an/aus */
    UBYTE WorkName[FILENAME_SIZE];         /* Zwischenspeicher
    für Druckername */
    BYTE RowSizeChange;
    BYTE ColumnSizeChange;
    BYTE padding[14];
*/------------------------ENDE VERSION 2.1-------------------------*/
    UWORD PrintFlags,PrintMaxWidth,PrintMaxHeight;        /*
    weiter Einstell- */
    UBYTE PrintDensity,PrintXOffset;               /* möglichkeiten für
    Printer */
    UWORD wb_Width,wb_Height,wb_Depth;  /* Breite, Höhe, Tiefe der
    Workbench */
    UBYTE ext_size;                     /* die Größe eventueller
    Erweiterungen */
};
```

Listing 41. Die Preferences-Struktur enthält alle Grundeinstellungen Ihrer Workbench

Ihre Einstellungen: Preferences

Befassen wir uns gleich, wie versprochen, mit den Funktionen GetPrefs() und SetPrefs(). Die erste, hier nocheinmal der Aufruf:

```
GetPrefs(&p,s);
```

kopiert die Daten in den als ersten Parameter angegebenen Bereich. Der zweite Parameter gibt die Größe des zu kopierenden Bereichs an. Man braucht ja nicht immer die ganzen Informationen, daher findet man die wichtigsten auch ganz vorne in der Struktur. In »DMReq.c« verlangen wir dennoch die ganze Struktur und geben daher als ersten Parameter die Adresse einer Preferences-Struktur und als zweiten Parameter deren Größe an.

Falls Sie die auf der System-Diskette gespeicherten Werte lesen möchten, wenden Sie GetDefPrefs() an. Die Parame-

```
/*
\\      Klick.c - testet auf Doppelklick
*/
#include "TOOL.h"

VOID main()
{
    struct Window *win;
    ULONG class,S[2],M[2];
    USHORT code=0;
    SHORT n=0;

    if(!(OpenLib())){
        ModifyIDCMP((win=ACTIVE_WINDOW),MOUSEBUTTONS);
        win->Flags^=RMBTRAP;
        while(code!=MENUDOWN){
            Wait(1<<win->UserPort->mp_SigBit);
            class=GetMessage(win,&code,0,0);
            if((class==MOUSEBUTTONS)&&(code==SELECTDOWN)){
                CurrentTime(&S[n],&M[n]);
                if(n){
                    if(DoubleClick(S[0],M[0],S[1],M[1]))
                        printf("DoubleClick !!!\n");
                    else  printf("zu langsam (gähn)\n");
                }
                n=n?0:1;
            }
        }
        win->Flags^=RMBTRAP;
        ModifyIDCMP(win,0);
        CloseLib();
    }
}
```

Listing 42. Klick.c zeigt wie man auf einen Doppelklick testet

ter der beiden Funktionen sind gleich.

SetPrefs(&p,s,flag), der Name deutet darauf hin, ist das Gegenstück zu den beiden zuvor besprochenen Funktionen.

Die beiden ersten Parameter sind wieder gleich. Der dritte bestimmt, ob die Werte dauerhaft auf Diskette (TRUE) oder nur im Speicher (FALSE) geändert werden sollen.

Möchten Sie gerne wissen, ob Sie wieder einmal zu viel Zeit vor dem Amiga verbracht haben, dann könnte Ihnen vielleicht CurrentTime() helfen:

```
ULONG Sekunden,
Mikrosekunden;
...
CurrentTime(&Sekunden,
&Mikrosekunden);
```

Sie erhalten die Systemzeit in Sekunden und Mikrosekunden ab dem 01.01.1978. Natürlich gibt es zur Ermittlung von Uhrzeit und Datum wesentlich nützlichere Funktionen. Mittels CurrentTime() können Sie jedoch die notwendigen Parameter für

```
ULONG S1,S2,M1,M2;
BOOL ok;
ok=DoubleClick(S1,M1,S2,
M2);
```

Die Parameter S1, M1 und S2,M2 sind die Systemzeiten zweier Mausklicks. Die Funktion prüft mittels der Zeitdifferenz und des eingestellten Wertes in Preferences, ob es sich um einen Doppelklick handelte. Wenn ja, liefert sie TRUE, wenn nicht FALSE. Listing 42 demonstriert den Test auf Doppelklick.

Hier eine kurze Beschreibung: nachdem IDCMP- und Windowflags des gerade aktiven (CLI-) Windows auf das Arbeiten mit den beiden Maustasten vorbereitet sind, warten im der Hauptschleife auf MOUSEBUTTONS. Registriert GetMessage() im Feld Code der Message-Struktur einen Klick mit der rechten Maustaste, beenden wir das Programm, nach den entsprechenden Reinitialisierungen. Erreicht uns die Nachricht der Betätigung der linken Taste schalten wir die Variable n von 0 auf 1 oder umgekehrt, falls sie schon den Wert 1 hat. Entsprechend des Wertes in n schreiben wir die Systemzeit in S[0] und M[0] oder in S[1] und M[1]. Immer wenn n den Wert eins hat, die Maustaste wurde dann zum zweiten Mal betätigt, testen wir auf Doppelklick. Damit sind wir am Ende unserer kleinen Übersicht zur Intuition angelangt. Natürlich kann diese nicht vollständig sein. Sie sollten sie ausgiebig mit eigenen Experimenten ergänzen.

Wenn Sie die bisher besprochenen Gebiete Intuition und Grafik beherrschen, sind Sie sicher auch in der Lage, gute Programme schreiben. Oft möchte man aber Daten auf Diskette sichern und dieselben von Diskette lesen. Dazu ist es unter Umständen notwendig, in bestimmte Verzeichnisse »hinabzutauchen«. Verzeichnisse müssen gelistet werden. Ebensowichtig ist die Pflege von Dateien und Verzeichnissen.

Von Daten und Disketten

Die notwendigen Funktionen müssen wir uns natürlich nicht selbst erarbeiten, sie sind bereits in der Systembibliothek von AmigaDOS enthalten. Die wichtigsten davon besprechen wir im Anschluß und erklären sie, – soweit nötig – an einem Beispiel.

Voraussetzung für die Benutzung der DOS-Routinen ist das Einbinden des speziellen Headerfiles »libraries/dos.h«. Dort sind unter anderem wichtige Strukturen für das Arbeiten mit Amiga-DOS definiert. Wir beschäftigen uns besonders mit den folgenden: »FileInfoBlock« und »InfoData«, definiert in dos.h, sowie FileHandle und FileLock aus dosextens.h. Das Headerfile dosextens.h wird von dos.h automatisch eingebunden, ist aber deshalb nicht weniger interessant.

Das Öffnen des DOS-Libraries und die Initialisierung der globalen Variablen DosBase erfolgt, wie ganz zu Beginn erwähnt, bereits im Startup-Programm. Natürlich ist es auch möglich, DosBase explizit in bekannter Weise zu initialisieren. Erweitern Sie doch einfach die Tool.h-Funktionen OpenLib() und CloseLib() mit den Funktionen zum Öffnen und Schließen der DOS-Library.

Beginnen wir mit den Funktionen, die mit der Adresse der FileHandle-Struktur arbeiten. Diese Struktur, wir bezeichnen den entsprechenden Pointer in den folgenden Funktionsaufrufen der Einfachheit halber als »FH«, steht stellvertretend für eine offene Datei oder ein offenes Device. Die Funktion Open (name,modus) öffnet die Datei »name« und liefert einen Zeiger auf die FileHandle-Struktur:

```
FH=Open(name,modus);
```

Der Parameter modus kann hierbei einen der symbolischen Werte »MODE_NEWFILE« oder »MODE_OLDFILE« annehmen. Als Parameter name kann auch ein Device-Name dienen. Sie erinnern sich, zu Anfang haben wir mit der Open()-Funktion das Gerät

Die Fehlermeldungen von IOErr()

FNr	Bedeutung
103	kein freier Speicher
105	zu viele (mehr als 20) aktive CLI-Prozesse
120	Argumentenliste zu lang oder fehlerhaft
121	kein ausführbares Programm
122	Fehler in einem residenten Library
201	Verzeichnis fehlerhaft
202	Objekt ist bereits geöffnet
203	es existiert bereits ein Objekt mit diesem Namen
204	Verzeichnis nicht gefunden
205	File nicht gefunden
206	fehlerhafter Stream-Name
207	Objekt zu groß
209	unbekanntes (unmögliches) Kommando
210	fehlerhafter Dateiname
211	fehlerhafte FileLock-Struktur
212	Objekt-Typ fehlerhaft
213	Diskettenerkennung negativ
214	Diskette ist schreibgeschützt
215	Umbenennung hier nicht möglich
216	Verzeichnis nicht leer
217	zu viele Levels (?)
218	Device nicht gefunden/eingebunden
219	feherhafte Positionsangabe
220	Kommentar zu lang
221	Diskette ist voll
222	Objekt vor löschen geschützt
223	Objekt vor überschreiben geschützt
224	Objekt vor lesen geschützt
225	keine DOS-Diskette
226	keine Diskette im Laufwerk
232	keine weiteren Einträge bei ExNext()

Tabelle 17. Die Funktion IoErr() liefert bei einem I/O-Fehler eine Nummer als Fehlermeldung

```
/*
\\      TYPE.c - ein File als Text listen
*/
#include "TOOL.h"

VOID main(argc,argv)
    SHORT argc;
    STRPTR argv[];
{
    UBYTE *buffer;
    LONG i,size;

    if((argc!=2)||(*argv[1]=='?')){
        printf("%c\033[3mAufruf:\033[0m%s Filename\n",7,argv[0]);
        exit(0);
    }
    if(buffer=GetFile(argv[1],&size)){
        printf("%c\033[4mliste %s ...\033[0m\n\n",12,argv[1]);
        for(i=0;i<size;i++) putchar(buffer[i]);
        FreeFile(buffer,size);
    }
    else printf("%cKann %s nicht listen!\n",7,argv[1]);
}
```

Listing 43. Eine Einsatzmöglichkeit für GetFile() – ein TYPE-Kommando

```
/*
\\      SavePlane.c - Speicherbereiche mittels PutFile() sichern
*/
#include "TOOL.h"

error(Meldung)
    STRPTR Meldung;
{
    if(Meldung) printf("Fehler beim %s!\n",Meldung);
    CloseLib();
}

VOID main()
{
    struct Window *win;
    struct Screen *scr;
    UBYTE *plane,*buffer;
    SHORT sw,sh,ww,wh;
    LONG i,size;

    if(OpenLib()) error("Öffnen der Libs's");
    scr  =ACTIVE_SCREEN;
    plane=scr->RastPort.BitMap->Planes[0];
    sw   =scr->Width;
    sh   =scr->Height;
    win  =ACTIVE_WINDOW;
    ww   =win->Width;
    wh   =win->Height;
    if(PutFile("RAM:plane",plane,sw*sh/8)) error("Speichern");
    if((ww>50)&&(wh>50)){
        SizeWindow(win,-20,-20);
        MoveWindow(win,20,20);
    }
    if(!(buffer=GetFile("RAM:plane",&size))) error("Laden");
    for(i=0;i<size;i++) plane[i]=buffer[i];
    FreeFile(buffer,size);
    RefreshWindowFrame(win);
    error(0);
}
```

Listing 44. Die Tool.h-Funktion PutFile() und Aktion: Speichern einer BitPlane

»CON«: angesprochen und so auf einfachste Weise ein System geöffnet.

Im Fehlerfalle liefert Open() anstelle des Zeigers FH den Wert null. Welcher Fehler tatsächlich auftrat, erfahren wir nach einem Aufruf der Funktion IoErr(). Sie kehrt mit einer DOS-Fehlermeldung zurück. Es handelt sich dabei um eine der in Tabelle 17 aufgelisteten Nummern.

Was kann man mit dem von Open() gelieferten Zeiger anfangen? Man könnte beispielsweise das File mit Close(FH) gleich wieder schließen, aber das ist wohl nicht Sinn der Übung...

Die Daten eines Files lesen ist schon etwas sinnvoller. Read(FH&puffer,(int)bytes) hilft uns dabei. Argument »puffer« verweist auf einen Speicherbereich von mindestens der Größe »bytes«, denn dieser dritte Parameter gibt an, wieviele Bytes Read() lesen soll. Der Rückgabewert der Funktion ist die Anzahl der tatsächlich gelesenen Bytes. Stimmt diese nicht mit der gewünschten Anzahl überein, ist das Dateiende erreicht. Ein Wert gleich -1 deutet auf einen Fehler hin.

Ähnlich arbeitet Write (FH, &puffer,(int)bytes). Eine Anzahl von »bytes« Bytes werden aus dem angegebenen Puffer in das entsprechende File geschrieben. Die Funktion liefert die Anzahl der tatsächlich geschriebenen Bytes. Im Fehlerfalle liefert sie ebenfalls -1.

Die besprochenen DOS-Routinen finden alle in ein Tool.h-Funktion Verwendung:

```
STRPTR file;
LONG size;
UBYTE *buffer;
...
buffer=GetFile(file,
&size);
```

Der Name läßt erahnen: die Funktion lädt ein File namens »file« in den Arbeitsspeicher, Returnwert ist die Adresse ab der das File im Speicher zu finden ist oder, falls ein Fehler auftrat null.

Nach dem Aufruf enthält die Variable »size«, deren Adresse beim Aufruf der Funktion als Parameter dient, die Größe des Buffers.

GetFile() alloziert den zur Aufnahme des Files notwendigen Speicher selbst. Für die Freigabe des Speichers müssen wir selbst sorgen. Wir haben dafür ein Macro definiert:

```
FreeFile(buffer,size);
```

Parameter sind die Adresse des Buffers und die Größe des Files, also genau die Werte, die GetFile() liefert. Listing 43 demonstriert eine Einsatzmöglichkeit für GetFile() und FreeFile(). Das kleine Programm listet ein File als Text. Nach Prüfung des Kommandoparameters laden wir mit GetFile() das gewünschte File. In der folgenden for-Schleife gibt putchar() jeweils ein Zeichen des Textes aus. Danach gibt FreeFile() den Speicher wieder frei.

Das Gegenstück zu GetFile() finden Sie ebenfalls in Listing 1:

```
Fehler=PutFile(file,
buffer,size);
```

Die Parameter sind:
file – ein STRPTR auf den Filenamen;
buffer – die Adresse des Speicherbereichs, den die Funktion speichern soll und
size, die Größe dieses Bereichs. Falls PutFile() den Bereich nicht sichern konnte, liefert die Funktion den Wert 1, ansonsten 0.

Auch hier ein Beispiel. Starten Sie »SavePlane« (Listing 44) vom CLI aus. Nach Ermittlung einiger Werte vom aktiven Screen und aktiven Window, speichern wir die unterste Bit-Plane in einem File namens »plane« im RAM: ab. Anschließend verändern wir Größe und Position des CLI-Windows, verändern also das Aussehen des Bildschirms. GetFile() läd das File wieder. Wir kopieren den geladenen Bereich zurück in Plane[0]. Und siehe da: das Bild des Windows erscheint an der ursprünglichen Position.

Die Haupt-Anwendung von GetFile() und PutFile() wird jedoch das Laden und – vielleicht nach einer Änderung – das Speichern von Files sein.

Datenübertragung vereinfacht

Möchte man eine bestimmte Position innerhalb eines Files lesen oder beschreiben, muß man den Schreib-Lesezeiger auf die betreffende Stelle positionieren. Eine Aufgabe für Seek(FH,(int)bytes,(int)modus). Hiermit sind wir in der Lage, den Zeiger, von der mit »modus« beschriebenen Position, »bytes« weit zu bewegen. Es existieren drei Modi: »OFFSET_BEGINNING«, »OFFSET_CURRENT« und »OFFSET_END«. Um beispielsweise ans Ende einer Datei zu gelangen, könnte der Aufruf so aussehen:

```
ap=Seek(FH,0,
OFFSET_END);
```

Die beiden Funktionen Input() und Output() liefern die Adresse der FileHandle-Struktur der Standardein- beziehungsweise -Ausgabedatei. Dies könnte zum Beispiel ein CLI-Task sein. Ob dies tatsächlich so ist, kann IsInteractive (FH) ermitteln. Die Funktion liefert »TRUE« wenn FH auf ein virtuelles Terminal verweist. WaitForChar(FH,t) erkennt, ob in einem virtuellen Terminal, in der angegebenen Zeit t (in Mikrosekunden), ein Zeichen eingegeben wurde und kehrt bei positivem Ergebnis mit »TRUE« zurück. Man erhält das Zeichen mit Read().

Eine Datei läßt sich neben der FileHandle-Struktur auch über die FileLock-Struktur identifizieren. Die letztere über-

```
struct InfoData {
        LONG id_NumSoftErrors;    /* Anzahl der registrierten
        Diskettenfehler */
        LONG id_UnitNumber;   /* Laufwerknummer. 0=DF0:, 1=DF1: usw,
         -1=RAM: */
        LONG id_DiskState; /* Diskstatus (validating,validated,
        writeprotect) */
        LONG id_NumBlocks;       /* Anzahl der zur Verfügung
        stehenden Blöcke */
        LONG id_NumBlocksUsed;                /* Anzahl der
        belegten Blöcke */
        LONG id_BytesPerBlock;                       /* Blocklänge
        in Bytes */
        LONG id_DiskType;         /* Disktype ('BAD\0','DOS\0','NDOS',
        'KICK') */
        BPTR id_VolumeNode;        /* Pointer auf Diskettenname
        (BCPL-String) */
        LONG id_InUse;                        /* wenn nicht in
        Gebrauch == 0 */
};
```

Listing 45. Die InfoData-Struktur enthält viele interessante Informationen über die Diskette in einem Laufwerk

```
struct FileInfoBlock {
        LONG fib_DiskKey;                /* Disk-Nummer (DOS-intern) */
        LONG fib_DirEntryType;          /* Eintrag > 0 ?
        Verzeichnis : Datei */
        char fib_FileName[108];            /* Filenamen, zur Zeit
        30 Zeichen */
        LONG fib_Protection;/* Protection-Bit - siehe CLI-Kommando
        Protect */
        LONG fib_EntryType;                 /* Type des Eintrag
        (DOS-intern) */
        LONG fib_Size;                         /* Größe einer Datei in
        Bytes */
        LONG fib_NumBlocks;                 /* Anzahl der belegten
        Blöcke */
        struct DateStamp fib_Date;             /* LONG Days/Minutes/
        Ticks */
        char fib_Comment[116];                        /* Kommentar */
};
```

Listing 46. Alle Daten eines Files oder eines Verzeichnisses sind in der FileInfoBlock-Struktur zusammengefaßt

```
/*
\\     FileInfo.c - liest FileInfoBlock aus
*/
#include "TOOL.h"

struct FileInfoBlock *info;
struct FileLock *lock;

VOID error(Meldung)
    STRPTR Meldung;
{
    if(Meldung) printf("Kein %s!?!\n",Meldung);
    if(lock)    UnLock(lock);
    if(info)    FreeMem(info,sizeof(struct FileInfoBlock));
    exit(0);
}

VOID main(argc,argv)
    SHORT argc;
    STRPTR argv[];
{
    if((argc!=2)||(*argv[1]=='?'))
        error("Argument (Dir-/Filename)");
    if(!(info=AllocMem(sizeof(struct FileInfoBlock),MEMF_CLEAR)))
        error("Speicher");
    if(!(lock=(struct FileLock*)Lock(argv[1],ACCESS_READ)))
        error(argv[1]);
    if(!(Examine(lock,info)))
        error("Info");
    printf("Name : %s\n",info->fib_FileName);
    printf("Type : ");
    info->fib_DirEntryType>0 ?
        printf("Verzeichnis\n") :
        printf("Datei\nGröße: %ld Bytes\n",info->fib_Size);
    error(0);
}
```

Listing 47. FileInfo gibt einige Werte der FileInfoBlock-Struktur eines beliebigen Eintrags aus

wacht aber noch die Schreib- und Lesezugriffe auf diese Datei. Die Funktion Lock (name,(int)modus) ermittelt die Adresse der FileLock-Struktur (FL) der mit Namen angegebenen Datei und übergibt den gewünschten Zugriffsmodus. Die Angabe von »ACCESS__READ« als Modus ermöglicht den Lesezugriff. Andere Tasks haben weiterhin Lesezugriff auf die genannte Datei. Einen solchen »Lock« kann man mit DupLock (FL) kopieren. Die Funktion liefert einen Zeiger auf die duplizierte FileLock-Struktur. Schreibend kann immer nur ein Task auf die Datei zugreifen. Modus »ACCESS__WRITE« sperrt alle anderen Schreibzugriffe. Lock liefert im Fehlerfalle »FALSE«.

Die Funktion UnLock(FL) gibt den mit Lock() erzeugten »Lock« wieder frei. Eine völlig neue FileLock-Struktur liefert CreateDir(name). Die Funktion erzeugt, wie das CLI-Kommando MakeDir, ein neues Unterverzeichnis im aktuellen Diskettenverzeichnis. Im Fehlerfalle ist der Returnwert gleich null.

IoErr() gibt über einen aufgetretenen Fehler näher Auskunft. CurrentDir(FL) erinnert ebenfalls an ein CLI-Kommando. Die Funktion erklärt das mit FL beschriebene Unterverzeichnis zum aktuellen Unterverzeichnis. Als Rückgabewert erhalten wir die Adresse der FileLock-Struktur des vorherigen aktuellen Unterverzeichnisses. Mit Hilfe dieser Routine ist es möglich, in einen Verzeichnisast des Dateisystems zu gelangen. Den umgekehrten Weg ermöglicht AmigaDOS mit der Funktion ParentDir(FL), welche die FileLock-Struktur des übergeordneten Verzeichnisses ermittelt. Ist der Returnwert gleich null, befindet sich das mit FL beschriebene File (oder Verzeichnis) im Ursprungsverzeichnis.

Gut informiert mit DOS-Strukturen

Die Funktion Info(FL,ID) initialisiert die als zweiten Parameter übergebene InfoData-Struktur (Listing 45). Diese enthält interessante Informationen über die Diskette auf der sich das File (oder Verzeichnis) der angegebenen FileLock-Struktur befindet.

Beachten Sie hierbei, daß die InfoData-Struktur im Speicher ab einer Langwort-Adresse beginnen muß. Reservieren Sie am besten den notwendigen Speicherplatz mit AllocMem(), da diese Funktion in jedem Fall einen Speicherbereich, beginnend ab einem Langwort, alloziert (siehe auch Listing 47).

Die Tool.h-Funktion GetLW() zeigt die Vorgehensweise. Sie ermittelt das aktuelle Laufwerk und liefert die folgenden Werte:

```
-1=Fehler
 0=RAM:
 1=DF0:
 2=DF1: etc.
```

Probleme beim Auslesen bereitet höchstens der »BCPL«-Pointer (BPTR). Rechnen Sie ihn mit dem in dos.h definierten Macro BADDR(BPTR) in einen normalen C-Pointer um. Neben BPTR begegnen Sie im AmigaDOS immer wieder dem Datentyp BSTR. Es handelt sich um BCPL-Strings. Das erste Byte eines BSTR beinhaltet die Länge des nicht nullterminierten Strings. Hier ein Beispiel für den String »SYS« aus den »Rom Kernal Reference Manual« Seite D-169:

```
s[0]=3 s[1]=S s[2]=
Y s[3]=S
```

AmigaDOS informiert nach Wunsch auch über ein Verzeichnis oder sogar über ein einzelnes File. Diese speziellen Informationen sind in der File InfoBlock-Struktur vereint. Um sie auszulesen steht uns die Funktion Examine(FL,FIB) zur Verfügung.

Funktionell entspricht Examine() der Funktion Info(). Natürlich initialisiert Examine() nicht die InfoData-Struktur, sondern die FileInfoBlock-Struktur (Listing 46), auf die wir mittels des als zweiten Parameter übergebenen Zeigers verweisen. Auch für diese Struktur reservieren wir den notwendigen Speicherbereich zur Sicherheit mit AllocMem().

Wie Sie ihr entnehmen können, sind darin wertvolle Informationen verborgen. Daher soll an dieser Stelle ein Beispielprogramm das Beschriebene verdeutlichen. Listing 47 demonstriert die Handhabung der FileInfoBlock-Struktur.

Wegen der Ähnlichkeit der Funktionen Info() und Examine() können Sie es ohne großen Aufwand zum Auslesen der In-

foData-Struktur umschreiben. FileInfo (Listing 47) erwartet beim Aufruf einen File- oder Verzeichnis-Namen als Kommandoparameter. Der vollständige Name - den Unterschied bemerken Sie zum Beispiel beim Aufruf mit »DF0:« als Argument - und der Typ des Eintrags werden gelistet. Handelt es sich um eine Datei, so gibt das Programm noch deren Größe in Bytes aus. FileInfo.c ist relativ kurz und auch einfach aufgebaut, so daß wir auf eine weitere Besprechung verzichten können.

Als weiteres Beispiel sei die Exists(filename) aufgeführt. Diese Funktion prüft, ob ein Eintrag vorhanden ist. Ist der Returnwert kleiner 0, so handelt es sich bei dem als Parameter mitgegebenen Eintrag um ein Verzeichnis. Ein Returnwert größer 0 deutet auf ein File hin. Der Wert entspricht in diesem Fall der Größe des Files in Byte. Im Fehlerfalle liefert Exists() den Wert 0. Sie finden die Funktion in Tool.h unter DOS-Tools definiert.

Eine weitere sehr interessante Funktion ist ExNext(FL,FIB). Sie ermöglicht in Zusammenarbeit mit Examine() die Untersuchung eines kompletten Verzeichnisses. Gehen Sie wie folgt vor: zuerst erfolgt der Aufruf von Examine() mit dem Lock des gewünschten Verzeichnisses. Zeiger auf FileLock- und neu erstellte FileInfoBlock-Struktur übergeben Sie ExNext(). Bei der Rückkehr der Funktion sind beide Parameter auf den ersten beziehungsweise nächsten Eintrag im Verzeichnis ausgerichtet.

Untersuchen Sie den Eintrag anhand der FileInfoBlock-Struktur, geben Sie eventuell Namen und Größe auf den Bildschirm aus, etc...

Rufen Sie ExNext() nun solange auf, bis keine Einträge mehr vorliegen. Der Returnwert der Funktion ist dann gleich null. Aber Vorsicht, auch wenn ein Fehler auftrat, liefert ExNext() Null.

Klarheit verschafft wieder IoErr(). Ist der letzte Eintrag erreicht, lautet die IoErr()-Fehlermeldung »ERROR_NO_MORE_ENTRIES«.

Directory ohne Probleme

Eine »klassische« Anwendung des geschilderten Ablaufs ist das DIR-Kommando, wie in Tool.h definiert. Wir sollten ReadDir() (Listing 1) etwas genauer beleuchten. Zuerst der Aufruf:

```
STRPTR d;
USHORT a;
SHORT e,f;
UBYTE n[N][FLEN];
LONG s[N];
...
e=ReadDir(d,a,f,n,s);
```

Beginnen wir von vorne:

e - der Returnwert - entspricht der Anzahl der gelesenen Einträge, im Fehlerfalle jedoch -1.

Der erste Parameter - **d** - ist ein Pointer auf den Namen des zu listenden Verzeichnisses. Geben Sie hier NULL an, liest ReadDir das aktuelle Verzeichnis.

Parameter **a** ist die maximale Anzahl der zu lesenden Einträge.

Die Namen der Einträge werden in einem Array gebunden. Sollte ReadDir weniger Einträge finden als unter »a« gefordert, werden alle überzählige Zeichenketten in dem Array mit einem Nullbyte initialisiert.

Das **f** steht für die beiden Tool.h-Flags DIRS und FILES. Setzt man DIRS, so liefert ReadDir() nur Verzeichnisnamen. Möchte man nur Filenamen, setzt man FILES. Um ein komplettes Directory zu lesen, verknüpfen Sie beide Flags mit OR:

```
DIRS/FILES
```

Wie schon erwähnt schreibt ReadDir() die Namen der Einträge in ein Array, in unserem Beispiel n[N][FLEN]. N ist dabei ein Wert, der mindestens so groß wie »a«, der Anzahl der erwarteten Einträge, sein sollte. FLEN ist ein Tool.h-Symbol, es steht für die Länge eines Filenamens (31 Zeichen mit abschließender 0).

Falls erwünscht, liest die Funktion auch die Filegröße. Sie erwartet dann die Adresse eines LONG-Arrays mit N Elementen. N muß wieder mindestens die Größe von »a« haben.

Handelt es sich bei dem Eintrag um ein Verzeichnis schreibt ReadDir() an die entsprechende Stelle im Filelängen-Array eine -1.

Benötigt man die Filelänge nicht, gibt man als letzten Parameter 0 an. Zur Verdeutlichung haben wir in Listing 48 einige Beispielaufrufe und in Listing 49 gleich eine Anwendung formuliert. ReadDir() erlaubt es, wie DIR.c zeigt, mit gerade 20 Zeilen Quellcode ein kleines DIR-Kommano zu formulieren.

Rufen Sie das Programm mit einem Directory-Namen als Kommandoparameter auf, liest ReadDir() dieses Verzeichnis. Im Rumpf der for-Schleife geben wir es auf dem Bildschirm aus. DIR listet dabei die Unterverzeichnisse revers.

Trat ein beim Einlesen ein Fehler auf brechen wir das Programm mit einem Bildschirmblitz und einer entsprechenden Fehlermeldung ab.

Die Routine selbst ist leicht zu verstehen, wir haben weiter oben besprochen, wie das Lesen eines Verzeichnisses generell aussieht. Merkwürdig mag Ihnen vielleicht die Verwendung von »TYPE«, »NAME« und »SIZE« vorkommen. Das sind nur Symbole, die wir in Tool.h definiert haben. Sie ersparen uns und Ihnen Schreibarbeit und greifen einfach auf entsprechende Komponenten der FileInfoBlock-Struktur zu.

Widmen wir uns zum Schluß noch einigen DOS-Funktionen, die fast wie CLI-Kommandos arbeiten.

Wir kommen dabei ohne große Erklärungen aus, dann die Funktionen sind sehr einfach zu handhaben: DeleteFile(name) löscht den Eintrag mit dem Namen »name«:

```
BOOL ok;
ok=DeleteFile("Test");
```

Rename(alt,neu) gibt dem Eintrag »alt« den neuen Namen »neu«. SetComment(name,kommentar) hängt einer Datei oder einem Verzeichnis einen Kommentar von maximal 116 Zeichen an. SetProtection(name,maske) kommt dem CLI-Kommando Protect gleich. In der Version 1.2 sind vier Protect-Bits erlaubt, es sind die vier untersten eines 32 Bit-Wertes. Dabei bedeutet:

Bit 0 - Schutz gegen Löschen;

Bit 1 - Datei nicht ausführbar;

Bit 2 - gegen Überschreiben geschützt (unbeschreiblich);

Bit 3 - verbietet das Lesen der Datei.

```
/*
\\    Aufrufe der TOOL.h-Funktion ReadDir()
*/
 #include "TOOL.h"

{
....
....
SHORT Einträge,Flags;
UBYTE Namen))Ä50))Ü))ÄFLEN))Ü;
LONG Size))Ä50))Ü;
...
/* maximal 50 Einträge, Files und Directories, lesen und
im Array Namen ablegen, Filelängen in Size ablegen: */
 Einträge=ReadDir("DF0:",50,FILES))öDIRS,Namen,Size);
/* nur Unterverzeichnisse lesen, ohne Files, ohne Size */
Einträge=ReadDir("DF1:",20,DIRS,Namen,0);
/* nur Files lesen, ohne Unterverzeichnisse, mit Filegröße */
Einträge=ReadDir("SYS:",30,FILES,Namen,Size);
...
...
}
```

Listing 48. Einige Beispielaufrufe der ReadDir()-Funktion

```
/*
\\ DIR - Kommando mit der TOOL.h-Funktion ReadDir()
*/
 #include "TOOL.h"
VOID main(argc,argv)
 SHORT argc;
 STRPTR argv))Ä))Ü;
{
 SHORT i,Eintr,Flags;
 UBYTE Namen))Ä50))Ü))ÄFLEN))Ü;
 LONG Size))Ä50))Ü;
 Eintr=ReadDir(argv))Ä1))Ü,50,FILES))öDIRS,Namen,Size);
 if(Eintr>=0){
 for(i=0;i<Eintr;i++){
 if(Size))Äi))Ü<0) printf("\033))Ä7m%s\033))Ä0m\n",Namen))Äi))Ü);
 else printf("%s\r\t\t\t\t%d\n",Namen))Äi))Ü,Size))Äi))Ü);
 }
}
else printf("%cKann %s nicht listen!\n",7,argv))Ä1))Ü);
}
```

Listing 49. Ein DIR-Kommando, superkurz dank Tool.h

Alle Funktionen dieses »Schnelldurchlaufs« liefern bei ordnungsgemäßer Funktion den Bool'schen Wert »TRUE«, ansonsten »FALSE«.

Wenn Sie mit den aufgeführten Funktionen lieber keine eigenen CLI-Kommandos schreiben möchten, so können Sie mit Execute(anweisung,FH_I, FH_O) doch wenigstens Existierende vom eigenen Programm aus starten.

Die Parameter zwei und drei sind die Adressen von File-Handle-Strukturen des Inputs und des Outputs.

Geben Sie für FH_I Null an, so wird der String »anweisung« ausgeführt. Alle Ausgaben der Anweisung erfolgen in FH_O. Null für FH_O nutzt den Standard-Output, zum Beispiel das aktive CLI.

Alle Befehle mit Execute

Wenn Sie für FH_I die Adresse der FileHandle-Struktur eines eigenen Windows übergeben, führt die Angabe eines Nullzeigers als Anweisungsstring zum Start ein neuen CLI's (siehe auch Listing 8). Voraussetzung für das korrekte Arbeiten der Execute()-Funktion ist das Vorhandensein des Programms RUN im Verzeichnis C:. Die mit Execute() aufgerufenen Programme laufen als Task, als wären Sie vom CLI aus mit RUN aufgerufen worden. Einige kleine »Probleme«, die es zu lösen gilt, sollen an dieser Stelle den Kurs abrunden:

- Schreiben Sie ein Programm das ständig das aktuelle Verzeichnis anzeigt – auch nach einem Diskettenwechsel.
- Erstellen Sie ein neues CLI-Kommando namens Print. Das Programm soll ein File, dessen Name als Kommandoparameter geliefert wird, ausdrucken.
- Ändern Sie Listing 44 so ab, daß der komplette aktuelle Screen als File abgelegt wird.
- Ändern Sie TYPE.c (Listing 43) so ab, daß Sonderzeichen unterdrückt werden. Der Anwender kann dann auch ausführbare Programme listen.
- Verändern Sie das aktive CLI-Window so, daß Sie ihre Compilier-Optionen in Zukunft über Pull-Down-Menü anwählen können.
- Schreiben Sie einige Programme, die wichtige CLI-Kommandos ersetzen können. Fassen Sie sie in einem Programm zusammen, welches Sie parallel zum CLI benutzen können. Die einzelnen Funkionen soll man über Pull-Down-Menüs wählen können.
- Entwickeln Sie ein Programm, welches erlaubt, einen beliebigen Ausschnitt des Workbenchscreens auszudrucken.
- Versuchen Sie den Workbenchscreen auf eine Bit Planes zu beschränken oder um BitPlanes zu erweitern.
- Schreiben sie eine Routine, mit der man den den Inhalt eines Windows mit den Cursortasten in vier Richtungen scrollen kann.
- Schreiben sie eine Routine, mit der man den den Inhalt eines Windows mit den Cursortasten in vier Richtungen rotieren kann. Der Inhalt darf also nicht verloren gehen. Was beispielsweise links verschwindet, soll rechts wieder erscheinen.
- Verändern Sie vom Programm aus den Mauszeiger. Das Programm soll beim Aufruf die Hintergrundfarbe des aktiven Screens verändern. Beim Verlassen muß natürlich die alte Farbe wieder hergestellt werden.
- Zeichnen Sie einige geometrische Figuren in ein Window. Kopieren Sie dessen Inhalt in eine BitMap und von dort in ein anderes Window.
- Definieren Sie mehrere Bool-Toggle-Gadgets. Ganz gleich, welches Gadget der Anwender bedient, es darf immer nur das zuletzt gewählte als aktiviert erscheinen.
- Schreiben Sie ein Programm, das Guru-Nummern in Klartext wandelt. Die Nummer sollte man in ein String-Gadget eintragen können.

Und zum Schluß noch für die Profis unter Ihnen:

- Alle professionellen Programme verfügen über einen File-Requester oder File-Window. Diese Routinen zeigen in einem Block das aktuelle Verzeichnis an. Paßt dieses von der Länge her nicht in den Block, kann man mittels eines Proportional-Gadgets die Anzeige nach oben und unten scrollen.

Mehrere Gadgets, zu Beispiel mit der Aufschrift »DF0:« oder »DF1:« erlauben es, mit einem Mausklick das Laufwerk zu wechseln. Ein Klick auf einen Verzeichnisnamen im Anzeigeblock und schon wird dieses Verzeichnis gelistet.

In String-Gadgets wird der Anwender ständig über den Pfadnamen und den Filenamen, den man ebenfalls bequem per Mausklick im Anzeigeblock wählt, informiert. Natürlich kann man dort auch die Namen über die Tastatur eingeben.

Mittels zwei weiteren Gadgets, mit der Aufschrift OK und CANCEL, kann der Anwender seine Wahl bestätigen oder abbrechen. Möchten Sie in Zukunft auf derartige Bedienungsfreundlichkeit in Ihren Programmen verzichten?

Falls Ihnen die eine oder die andere unserer Übungsaufgaben nicht lösbar erscheint, lesen Sie im betreffenden Kapitel noch einmal nach. Alle Aufgaben beziehen sich auf eine Aussage im Text.

»Verdauen« Sie den besprochenen Stoff – zugegeben, es war eine ganze Menge – und testen Sie einfach alle Routinen ausgiebig, innerhalb kürzester Zeit werden Sie sich fragen, wie Sie solange ohne sie ausgekommen sind. Wir hoffen, daß Ihnen mit unserem C-Kurs für Amiga-Programmierer den Einstieg in die Programmierung mit Betriebssystemroutinen gelingen wird und daß Ihnen, wie zu Anfang versprochen, unser Tool.h zu einem wertvollen Werkzeug geworden ist. Tabelle 18 stellt nochmals alle Funktionen und Makros und deren Aufgabe gegenüber.

Mit diesem Headerfile wird die Programmierung der – zugegeben relativ komplizierten – Intuitionfunktionen stark vereinfacht. Sie sparen sich die Initialisierung einiger Parameter, die auf Standardwerke gesetzt werden. Daneben erhalten Sie – wie Tabelle 18 zeigt – 29 neue hilfreiche Funktionen, die Ihnen das Leben mit Intuition + Co. erleichtern.

Wir wünschen Ihnen beim weiteren Arbeiten mit Ihrem Amiga viel Erfolg und vor allem viel Spaß!

(Arno Gölzer/so)

Tool.h-Funktionen zu Intuition und Grafik

Name	Aufgabe der Tool.h-Funktion
OpenLib()	für Intuition und Grafik benötigte Libraries öffnen
CloseLib()	die mit OpenLib() geöffneten Libraries schließen
GetScreen()	Screen öffnen
ACTIVE_SCREEN	Zeiger auf aktiven Screen (Makro)
GetWindow()	Window öffnen
ACTIVE_WINDOW	Zeiger auf aktives Window (Makro)
GetMessage()	Intuition-Message und verschiedene Werte lesen
Line()	Linie zeichnen
ILine()	Linie im JAM1))öCOMPLEMENT-Modus zeichnen
Block()	ausgefüllter Block zeichnen
IBlock()	Block im JAM1))öCOMPLEMENT-Modus zeichnen
IRast()	RastPort im JAM1))öCOMPLEMENT-Modus darstellen
GetPos()	Position des Grafikcursors im RastPort ermitteln
GetCol()	Ermittelt Farbwert und splittet ihn in R-G-B
Print()	Ausgabe eines Textes
EllFill()	ausgefüllte Ellipse zeichnen
CircFill()	Ausgefüllter Kreis zeichnen (Makro)
HrdCpy()	RastPort (oder Teil davon) in beliebiger Größe ausdrucken
GetGadget()	Gadget initialisieren, in Liste einfügen und darstellen
GetMenu()	Menu-Struktur initialisieren, in die Menu-Liste einfügen
GetItem()	MenuItem-Struktur initialisieren, in die Item-Liste einfügen
GetSItem()	SubItem initialisieren und in die SubItem-Liste einfügen
GetRequest()	frei definierbaren Requester (Dialogfeld) installieren
Exists()	Existenz eines Eintrags feststellen
GetFile()	File in Speicher laden
FreeFile()	Speicher für File freigeben (Makro)
PutFile()	Puffer speichern
GetLW()	die Nummer des aktuellen Laufwerks ermitteln
ReadDir()	Diskettenverzeichnis und Informationen lesen

Tabelle 18. Die Tool.h-Funktionen vereinfachen die Programmierung von Intuition und Grafik

Faszinierend: Maschinensprache
jedoch das Gerücht ver-
in Assembler sei ein Job nur für Bit-
Alchimisten. Lassen
Bockshorn jagen. Assem-
mierer kochen
Ende dieses Kurses von
sind auch Sie in der Lage, Pro-
mit Maschinensprache

Ein in Assembler geschriebenes Programm ist ungefähr 200mal schneller als ein Basic-Programm und oft genug noch 10mal schneller als die Lösung in C. Nun mag ja manche Leute das Tempo nicht beeindrucken, sie haben Zeit. Es gibt jedoch weitere Gründe, sich mit Maschinensprache zu befassen.

Ein Basic-Interpreter (oder ein C-Compiler) kratzt eigentlich nur an der Oberfläche eines Riesenpotentials von Möglichkeiten, die in einem Computer, beispielsweise dem Amiga, stecken. Will man mehr, dann muß man das der CPU (der Zentraleinheit) nur sagen – allerdings in ihrer Sprache, und das ist nun mal Assembler.

Noch ein Grund: Man sollte eigentlich immer die Sprache verwenden, die das jeweilige Problem mit minimalem Aufwand löst. Oft genug, sogar meistens, ist das nicht Assembler. Ich möchte sogar fast behaupten, je besser man Assembler kann, desto weniger braucht man diese Sprache. Ein Assembler-Programmierer weiß nämlich, was er mit welchen Befehlen, zum Beispiel in Basic, der CPU an Arbeit zumutet und kommt so zwangsläufig zu besseren Programmen. Denn das muß ich nun leider schon einmal andeuten, Assembler setzt gute Kenntnisse der Funktion eines Computers voraus. Aber zum Trost: Diese Kenntnisse erwirbt man am besten, wenn man Assembler lernt.

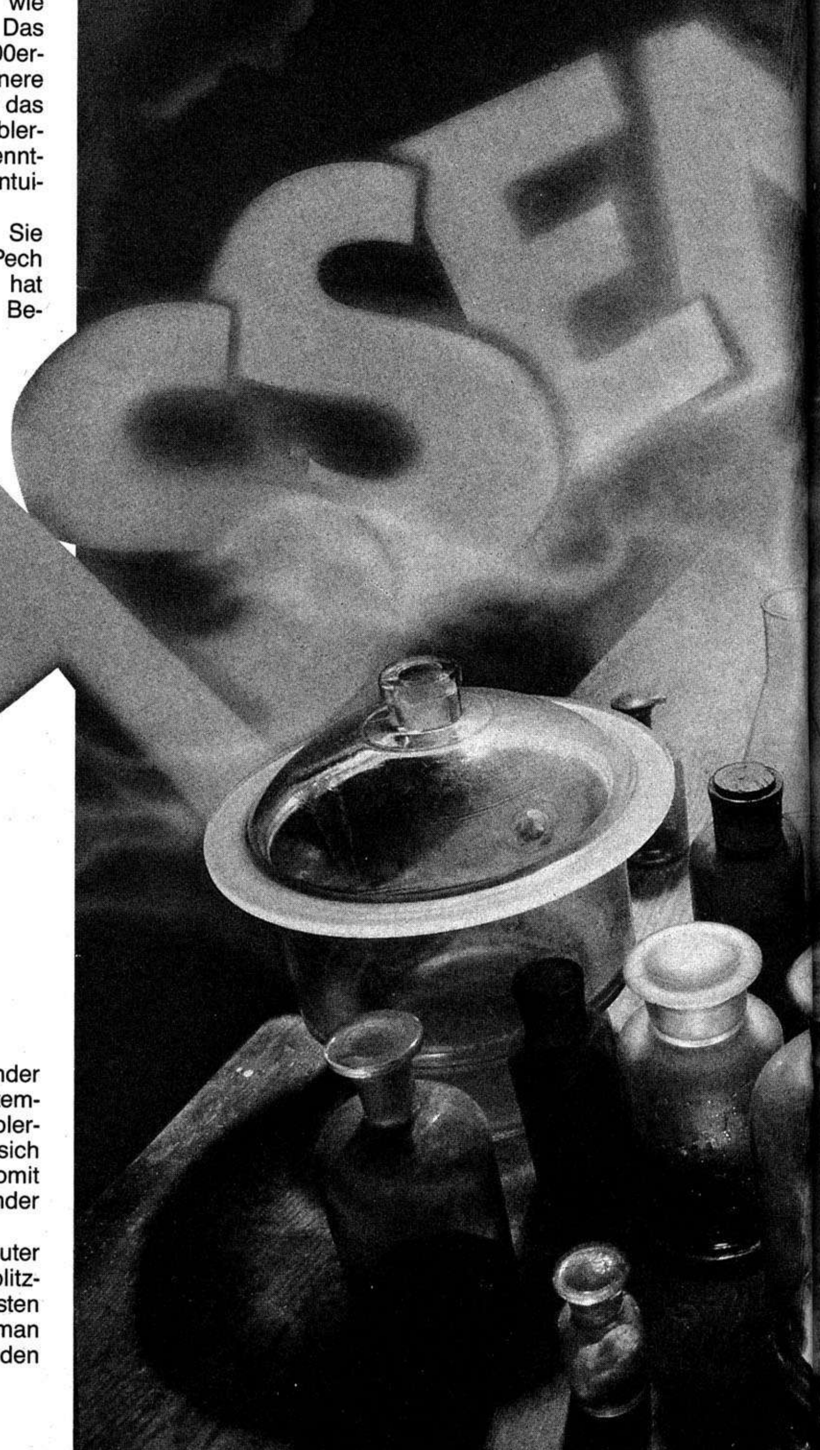
Untersucht man ein Programm, das in einer Hochsprache geschrieben wurde oder geschrieben werden soll, dann stellt man fest, daß es nur an einigen Stellen (meistens nur an einer Stelle) das Tempo-Problem gibt oder die passende Funktion fehlt. Dann sollte man auch nur diesen Teil in Assembler schreiben und ihn in die Hochsprache einbinden.

Eine neue Fremdsprache

Im Gegensatz zur höheren Mathematik, die schwieriger ist als die »niedrige«, ist es bei den Programmiersprachen eher umgekehrt. Die Sprachen weit weg von der Maschine nennt man Hochsprachen, in Assembler sind wir »ganz unten«. Auch wenn Sie später nur noch in den höheren Regionen schweben, Sie wissen, mit einer soliden Grundausbildung schwebt es sich leichter. Man fällt dann nicht so leicht herunter. Wir müssen jetzt aber in diese Niederungen hinabsteigen, denn was da unten los ist, müssen Sie schon wissen.

Wer eine Programmiersprache beherrscht und sie erfolgreich anwendet, der lernt auch jede andere. Die Sache ist nämlich ganz einfach. Entweder man kann programmieren oder man kann es nicht. Man muß ein paar neue »Vokabeln« lernen, aber die Grundlagen, wie Schleifen, Verzweigungen oder Unterprogramme wendet man wie üblich an. Solche Grundkenntnisse setzen wir für diesen Kurs voraus. Basic, C oder Pascal sollten Sie also schon können. Generell ist Assembler nicht schwieriger als Basic, nur etwas umständlicher. Zu diesen Umständlichkeiten gehört auch, daß Sie wissen müssen, wie der Computer funktioniert. Das betrifft nicht nur den 68000er-Prozessor (das ist das kleinere Problem), sondern primär das Betriebssystem. Assembler-Programmierung ohne Kenntnisse von Exec, DOS oder Intuition ist nicht möglich.

In einem Punkt haben Sie übrigens mit Ihrem Amiga Pech gehabt. Unsere Freundin hat zwar das komfortabelste Betriebssystem aller PCs, aber leider auch das komplizierteste.



Der Luxus, den der Anwender sieht, ist die Arbeit der Systemprogrammierer. Als Assembler-Programmierer dürfen Sie sich zu letzteren zählen, und somit schuften, damit der Anwender seine Freude hat.

Scheinbar ist ein Computer ein Rechengenie, das blitzschnell die kompliziertesten Aufgaben löst. Doch geht man der Sache einmal auf den

auf dem Amiga. Irgendwie hat sich
breitet, Programmieren
Magier und
Sie sich nicht ins
bler-Program-
auch nur mit Wasser. Am
Peter Wollschlaeger
gramme
aufzuwerten.

BLER

Grund, stößt man auf die schreckliche Wahrheit. Unser Wunderkind kann nicht bis Drei zählen, noch nicht einmal bis Zwei, es kennt gerade die Null und die Eins. Ursache ist, daß die elektrischen Schaltkreise, aus denen ein Computer besteht, nur zwei Zustände annehmen können, nämlich »Spannung da« oder »Spannung nicht da«.

Von Null bis Eins

Ein paar Hunderttausend dieser Schaltkreise bilden nun die CPU (Central Processing Unit, Zentraleinheit, praktisch das Herz des Computers), nochmals mehr als 8 Millionen davon (so Sie einen Mega-Amiga haben) sind der Speicher (das Gedächtnis) des Computers.

Ein Programm ist nichts weiter als ein bestimmter Zustand dieses Speichers. Da es nun höchst unpraktisch ist, ein Programm in der Art zu beschreiben »Transistor 1 leitet, Transistor 2 auch, Transistor 3 sperrt und so weiter«, kam man schnell auf eine Kurzschreibweise dieser Art: Der eine Zustand heißt 0 der andere 1.

Höher: Assembler

So kann man ein Programm doch schön kompakt schreiben, zum Beispiel als:

```
0101110011010101010l0
u.s.w.
```

Das gefällt Ihnen nicht? Nun, das ist die Maschinensprache, mehr nicht! Was Sie vielleicht schon erkannt haben: Dieses 0101011-Muster ist eine Zahl in dualer Schreibweise. Die Zahlen kann man umrechnen in Dezimal- oder Hexadezimal-Zahlen, das spart etwas Papier, es bleibt aber Maschinensprache.

Manche Leute behaupten nun, Assembler sei diese Maschinensprache. Gott sei Dank haben die Unrecht, das wäre ja schrecklich. Die armen Kerlchen, die die ersten Computer so programmiert haben, tun mir heute noch leid. Assembler ist die nächsthöhere Stufe und war einst der ganz große Fortschritt und viele Jahre lang auch die einzige Sprache überhaupt. Nun muß ich leider doch noch etwas ausholen.

Wenn Sie bei Bit nicht mehr zuerst an Bier denken, sind Sie schon Programmierer, OK... Ein Bit ist eine Speicherstelle, ein solcher Schaltkreis im Computer, der nur diese Zustände 0 oder 1 annehmen kann. Aus technischen Gründen hat man immer 8 Bit zusammengefaßt, diese 8 Bit nennt man ein Byte. Der Speicher eines Computers besteht aus Tausenden oder Millionen von Bytes. Damit man nun jedes Byte ansprechen kann, sind sie durchnumeriert. Diese Hausnummern der Bytes nennt man Adressen. Mit den 8 Bit eines Bytes lassen sich in dualer Schreibweise die Zahlen 00000000 bis 11111111 darstellen, dezimal ist das 0 bis 255.

Bewegen ist alles

In ein Byte (der Fachmann sagt, auf eine Adresse) kann ich nun eine solche Zahl hineinschreiben und sie wieder herauslesen. Die sogenannten Peripherie-Geräte wie der Bildschirm, die Tastatur oder ein Drucker sind nun mit einem Teil des Speichers (unseren Bytes) verbunden. Wenn ich auf die richtige Adresse eine Zahl schreibe, dann erzeugt sie eine Wirkung auf dem Bildschirm, wenn ich aus einer anderen Adresse etwas lese, dann kann das zum Beispiel eine Taste des Keyboards sein.

Folglich besteht ein Programm zum großen Teil daraus, Zahlen – man spricht auch von Daten – auf eine Adresse zu schreiben, von einer anderen zu lesen und ganz wesentlich, Daten von einer Adresse (zum Beispiel Tastatur) auf eine andere Adresse (zum Beispiel Bildschirm) zu kopieren.

Neben den Daten kennt so ein Computer auch Befehle, natürlich auch nur als 010101110, sprich als Zahlen.

Nehmen wir an, die Zahl 11111111 ist der Befehl »Kopiere« und wir wollen Daten von der Adresse 0000011 (dezimal 3)

auf die Adresse 00001001 (dezimal 9) kopieren, dann lautet dieses Programm in Maschinensprache:

```
11111111
00000011
00001001
```

In Assembler hingegen schreibt man dafür:

```
MOVE 3,9
```

»Move« heißt bewege, hier: bewege, was im Byte mit der Adresse 3 steht zum Byte mit der Adresse 9. Um gleich einen großen Denkfehler auszuschließen: Das Byte 3 bleibt unverändert, es wird nur in das Byte 9 kopiert. Sie haben recht, der Befehl müßte eigentlich COPY heißen, aber er heißt nun mal MOVE.

So, den Unterschied zwischen Assembler und Maschinensprache hätten wir, ist doch ein Fortschritt, oder?

Doch schon haben wir das nächste Problem. Der Begriff »Assembler« hat nämlich eine doppelte Bedeutung. Zum einen ist damit eine Programmiersprache gemeint, genauso wie zum Beispiel Basic oder Modula. Der Unterschied ist hauptsächlich, daß Assembler immer an eine bestimmte CPU (Hauptprozessor) gebunden ist. Es gibt zum Beispiel den 6502-Assembler, den 8088-Assembler und natürlich den 68000-Assembler, um den es hier geht. Die Sprache hat Befehle, wie alle anderen Sprachen auch, die tippen Sie auch einfach so ein, wie üblich.

Der große Unterschied zum Beispiel zu Basic ist dann nur, daß Sie danach nicht RUN geben können, sondern den Text erst assemblieren müssen. Genau das erledigt ein Programm, das dummerweise auch Assembler heißt. Dieses Programm übersetzt den Text in die Maschinensprache, also die 0101010-Folge, die die CPU letztendlich nur versteht.

Ganz oben: Hochsprachen

In einer Hochsprache, wie zum Beispiel C geben Sie auch nur Text ein, auch der muß übersetzt werden. Nur nennt man dann das Übersetzungsprogramm nicht Assembler, sondern Compiler. Das heißt, sowohl nach einem Assemblerals auch nach einem Compilerlauf entsteht ein Programm in Maschinensprache, das auf einem Computer ausgeführt werden kann. Über Größe und Schnelligkeit der Programme ist damit noch nichts gesagt.

Ganz anders sieht es bei einem Interpreter aus; der typischste Vertreter dieser Gattung ist wohl Basic. Auch hier geben Sie das Programm als Text ein. Vielleicht wird es nach der Eingabe noch etwas aufbereitet und komprimiert, aber es bleibt Text, der nicht die geringste Ähnlichkeit mit Maschinensprache hat.

Folglich kann der Computer ein Basic-Programm auch nicht ausführen. Diese Aufgabe übernimmt der Interpreter. Er liest den Basic-Text Zeichen für Zeichen und untersucht ihn auf Basic-Befehle. Findet er einen Basic-Befehl, so ruft er eine Routine auf, die den Befehl ausführt. Die Routine befindet sich natürlich als ausführbares Maschinenprogramm im Speicher. Sie übernimmt es auch, zu einem Basic-Befehl gehörige Daten (Parameter) im Basic zu suchen. Selbstverständlich ist auch der Interpreter selbst ein Programm in Maschinensprache. Alle schnellen Basic-Interpreter sind in Asembler geschrieben.

Assembler im Prinzip

Ja, wenn denn nun der Compiler genauso Maschinen-Code erzeugt, wie ein Assembler, dann sollte ich mir die Sache doch noch einmal genau überlegen.

In Pascal zum Beispiel schreibe ich einfach nur:

```
Write('Hallo')
```

und in Assembler tippe ich dafür (nur als Beispiel):

```
MOVE  #'H',4711
MOVE  #'a',4712
MOVE  #'l',4713
MOVE  #'l',4714
MOVE  #'o',4715
```

Demnach ist ein Assembler-Programm die Auflösung von zum Beispiel Pascal-Befehlen wie WRITE in viele Einzelbefehle. Man kann es auch anders sagen: Pascal kennt eine bestimmte Menge von Befehlen, aus denen der Compiler die passende Folge von Assembler-Befehlen erzeugt.

Tatsächlich ist jedes Assembler-Programm (in der noch nicht übersetzten Textform) immer länger als sein Äquivalent in einer Hochsprache. Nur wenn Sie einmal nach dem Assemblieren beziehungsweise dem Compilieren jeweils die Bytes des Codes zählen, dann ist ein Assembler-Programm drastisch kürzer und schon deshalb auch schneller. Das liegt daran, daß kein Compiler einen so kompakten Code generieren kann, wie es ein Assembler-Programmierer macht.

Letzterer weiß ja, was er will, er kann jede Befehlsfolge »maßschneidern«, ein Compiler hingegen muß Universallösungen einsetzen.

Ganz drastisch, im Tempo so bis zu Faktor 200, ist der Unterschied zu einem Basic-Interpreter. Dieser übersetzt – wie schon geschildert – erst während der Laufzeit, und dann immer nur einen Befehl. Das heißt, wenn in einer Schleife ein Befehl hundertmal wiederholt wird, dann wird er auch hundertmal übersetzt. In einem Assemblerprogramm hingegen ist der Befehl schon übersetzt.

Die typische Arbeitsfolge einer Programmentwicklung in Assembler sind Texteingabe, Assemblieren, Linken (kommt gleich) und Testen. Das sind Ihre Werkzeuge, und wie in jedem Handwerk kommt es darauf an, daß Sie mit den richtigen Werkzeugen arbeiten. Da gibt es nun leider eine große Auswahl, und die Prospekte der Hersteller versprechen alle viel. Ich möchte Ihnen hier einige Tips geben, die Sie bei der Auswahl beachten sollten. Ein Assembler ist ein Profi-Werkzeug, für das eine gute Dokumentation und Unterstützung durch den Hersteller oder Händler (Support) benötigt werden. Natürlich bekommen Sie ein (versehentlich) kopiertes Spielprogramm auch per »Trial and Error« zum Laufen. Dies auf einen Assembler anzuwenden, dürfte nur etwas für Leute mit sehr guten Nerven und unendlich viel Zeit sein. Es ist nämlich durchaus möglich, daß alle Programme in diesem Buch mit Ihrem speziellen Assembler nicht laufen, weil Ihr Assembler an einer Stelle einen Punkt verlangt, den meiner nicht braucht. Warum also knobeln, wenn alles im Handbuch steht.

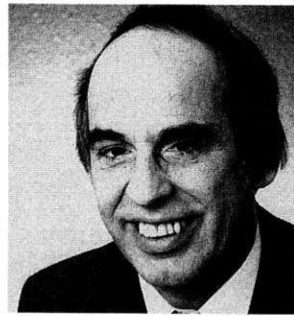
Einen Editor brauchen Sie für die Texteingabe und dessen Korrektur. Den Text nennt man Quell-Text (Source-Text). Üblicherweise werden Editor und Assembler zusammen verkauft. Sie können aber auch ohne weiteres Ihr gewohntes Textverarbeitungsprogramm nehmen, wenn Sie sich auf reinen Text (keine Formatier- und Steuerzeichen) beschränken. Auch ED (gehört zum Amiga) ist dafür brauchbar.

Ist der Text fertig (und auf der Diskette), starten Sie den Assembler, der dann mindestens wissen will, wie der File mit dem Quelltext (Source-File) heißt. Der Assembler erzeugt den Maschinen-Code (dieses 010101010), – auch Objekt-Code genannt – und legt diesen in dem Ziel-File (dem Objekt-File) auf der Diskette ab. Das kostet natürlich Zeit, und so erscheint es sinnvoll, auch »in memory« assemblieren zu können. Das bedeutet, der Assembler schreibt auf Wunsch den Code direkt in den Speicher, und man kann das Programm zu Testzwecken starten. Daß man es danach (oder vorher) auf die Diskette bringen kann, ist logisch.

Der Assembler kann Textmodule einbinden. Das ist sehr wichtig, denn Amiga-Programme benötigen immer die sehr zahlreichen Definitionen der sogenannten Libraries (Bibliotheken), die als Textmodule vorliegen. Kommt eine neue Workbench, gibt es auch neue Module, natürlich im Commodore-Format. Deshalb ist es extrem wichtig, daß ein Assembler dieses Format versteht.

Ausführlich werden Makros noch behandelt. Hier nur soviel: Makros tragen stark zur Rationalisierung der Arbeit bei und helfen, Fehler zu vermeiden. Auch gehören Makros zu den obengenannten Include-

Peter Wollschlaeger



...gehört noch zu der Technikergeneration, die digitale Schaltungen bis hin zum Computer mit einzelnen Transistoren entwickelte. Programmiert hat er damals nur nebenbei. Mit der Einführung der Mikroprozessoren verlagerte sich seine Tätigkeit zur Software. Durch ständige Weiterbildung und über 20 Jahre praktische Erfahrung als Systemprogrammierer wurde er zu einem Top-Experten für Mikrocomputer. Nebenberuflich arbeitet Peter Wollschlaeger als freier Mitarbeiter für verschiedene Zeitschriften und als Buch-Autor. Er bringt die journalistische Erfahrung aus weit über 100 Artikeln sowie mehreren Büchern für den Atari ST und Amiga mit. Dies befähigt ihn, auch schwierige Themen leicht verständlich und angenehm lesbar darzustellen.

Files, folglich gilt auch hier die Forderung, daß der Assembler Makros im Commodore-Format versteht.

Nun brauchen Sie den Linker, zu deutsch Binder. Der Binder hat zwei Aufgaben. Zum einen können Sie ein Programm in Module aufteilen, die Sie getrennt assemblieren und testen können (bei großen Programmen empfehlenswert).

Der Linker

Diese Module müssen Sie dann mit dem Linker zu einem Programm zusammenbinden. Der zweite Grund liegt beim Amiga selbst. Jedes Programm hat einen kleinen Vorspann, Header genannt, in dem zum Beispiel steht, wie groß das Programm ist. Ohne diese Information kann der Amiga das Programm nicht laden und starten. Folglich muß der Linker zumindest diesen Header mit Ihrem Programm binden. Es gibt aber auch Assembler, die das schon tun. Das erspart den zusätzlichen Linkerlauf, was man durchaus positiv sehen sollte.

Starten Sie nun Ihr Programm, gibt es drei Möglichkeiten: entweder es läuft, es läuft nicht oder es läuft falsch. Um den Bug (Programmierer-Slang für Fehler) zu finden, bieten sich viele Lösungen an. Die einfachste (und meist erfolgreichste) Methode ist ein tiefer Blick auf den Quelltext, kombiniert mit intensivem Nachdenken.

Wenn Sie aber wissen wollen, was das Programm an einer bestimmten Stelle tut oder welche Werte dann einige Variable haben, wird's schwierig. Möglich ist es, an diesen Stellen sozusagen ein »PRINT A,B« einzubauen, was aber in Assembler recht aufwendig ist, wie wir noch sehen werden (es gibt keinen Print-Befehl).

Praktischer ist es dann, einen sogenannten Debugger (Entwanzer, Fehler sind Wanzen!) einzusetzen. Das ist ein Programm, mit dem Sie Ihr Programm in Einzelschritten ablaufen lassen und sich an jeder Stelle die Werte der Variablen ansehen können.

Editor, Compiler, Linker und Debugger (soweit vorhanden) werden meistens im Paket angeboten (so sollte es sein). Häufig gehört dazu noch eine sogenannte Shell (eigene Benutzeroberfläche), die es Ihnen gestattet, zum Beispiel direkt vom Editor in die Shell zu wechseln, wo Sie dann den Linker aufrufen. Das ist, sofern man ohne RAM-Disk arbeitet, schneller als der Umweg über die Workbench beziehungsweise das CLI. Ansonsten hat eine gute Shell den Vorteil, daß sie sozusagen eine für die Programmierung maßgeschneiderte Workbench ist.

Eine ausführliche Vorstellung der beschriebenen Werkzeuge finden Sie in diesem Sonderheft ab Seite 6.

Generell läuft ein Programm in einem Computer nach der Methode »Fetch and Execute« (Holen und Ausführen). Die CPU holt sich aus dem Speicher einen Befehl und führt ihn aus. Danach holt sie sich automatisch den nächsten Befehl und führt diesen aus, und so weiter. Natürlich muß im Speicher etwas stehen, das die CPU holen und ausführen kann, und das nennt man dann Programm.

Ein Programm ist also nichts weiter als eine Folge von Bytes, die irgendwo im RAM oder ROM steht. Natürlich kann die CPU nicht wissen, wo das Programm im Speicher steht. Deshalb wird sie beim Start (Reset) per Hardware-Vorgabe (und später durch den Lader des GEMDOS) sozusagen mit der Nase auf eine Anfangsposition gestoßen. Ab diesem Augenblick holt sich die CPU immer ein Wort (das sind 2 Byte nebeneinander, also 16 Bit) aus dem Speicher und decodiert dieses Wort. Dabei kommt dann (hoffentlich) ein Befehl für die CPU heraus. Diesen Befehl arbeitet sie ab und holt dann das nächste Wort. Zu einem Befehl können Daten gehören. Beim Addier-Befehl zum Beispiel muß die CPU wissen, was addiert werden soll. Wieviel Datenworte zu einem Befehl gehören, ist auch im ersten Wort (dem Befehlswort) codiert.

Nur gerade Adressen erlaubt

Der gesamte Speicher ist Byte für Byte von Null bis zum Ende durchnumeriert, diese Nummern der Speicherplätze nennt man Adressen. Die CPU arbeitet immer nur mit diesen Adressen und führt dazu intern einen Zähler, der immer auf die aktuelle Adresse zeigt, bei der sie gerade ist. Diesen Zähler nennt man »Program Counter«, kurz PC.

Der 68000 kennt Befehle ohne Daten, die sind dann ein Wort lang, aber auch solche mit bis zu vier Datenworten. Das heißt, beim 68000 kann ein einziger Befehl mit seinen Daten bis zu 10 Byte (fünf Worte) belegen. Der 68000 ist zwar eine Byte-Maschine (er liest den Speicher Byte für Byte), jedoch muß jeder Befehl auf einer geraden Adresse (Wortgrenze) beginnen, andernfalls passiert Übles. Sie können (und müssen sogar) den PC verändern. Wenn nämlich ein Programm nicht nur einfach Befehl für Befehl abläuft, Sie also zum Beispiel ein GOTO benötigen, dann heißt das in Assembler zuerst einmal »GOTO Adresse«. Praktisch heißt das aber für die CPU »Setze PC = Adresse«. Wenn Sie da eine ungerade Adresse angeben, stürzt leider Ihr Programm ab.

In der Praxis tritt dieser Fehler auf, wenn Sie im Programm Daten definieren. Wenn Sie zum Beispiel den Text »Franz Meier« drucken wollen, müssen Sie irgendwo im Speicher eine Byte-Folge mit den ASCII-Codes dieser Zeichen haben. Folgt dann der Text »8000 München«, und Sie wollen diesen Text einmal alleine drucken, dann sollten Sie wissen, wie lang »Franz Meier« ist, damit »8000 München« auf einer geraden Adresse beginnt. Um diese Abzählerei zu ersparen, haben gute Assembler einen Befehl (EVEN oder CNOP), der Texte (oder Daten allgemein) auf eine gerade Adresse justiert. Ist die Adresse sowieso gerade, passiert nichts. Ein »Even« zuviel schadet also nichts, eines zu wenig dagegen sehr. So informiert, werden Sie also nie wieder vor einem Text den EVEN-Befehl (oder Gleichartiges) vergessen, wie das die anderen Anfänger tun, oder?

Das hexadezimale Zahlensystem ist in Assembler üblich (und sehr vorteilhaft), machen Sie sich bitte unbedingt mit diesem Zahlensystem vertraut. Hier ein Schnellkurs:

Die Basis ist nicht 10, wie im Zehner- oder Dezimalsystem, sondern 16. Für die fehlenden »Ziffern« von 10 bis 15 schreibt man A bis F. In dezimal sagt man für die Zahl 345 auch

5 Einer plus 4 Zehner + 3 Hunderter.

In hex ist die Basis 16. Die Folge wäre also nicht 1, 10, 100, 1000, sondern 1, 16, 256, 4096.

Sie wissen, F hat den Wert 15. Demnach ist FFFF:

15 * 4096 + 15 * 256 +15 * 16 + 15 * 1 = 65535

Listing 1 enthält ein kleines Programm in Amiga-Basic zum Üben. Wenn Sie eine Dezimalzahl eingeben, dann gibt sie das Programm in hex aus. Umgekehrt, geben Sie eine Hex-Zahl ein (erkenntlich am $ als erstes Zeichen), erhalten Sie deren Wert in dezimal.

Sozusagen noch eine Stufe tiefer (noch näher am Computer) ist das duale Zahlensystem. Hier ist die Basis 2, womit in diesem System nur die Ziffern 0 und 1 erlaubt sind. Am einfachsten kann man eine Dualzahl in dezimal umrechnen, indem man sich die Wertigkeit darüberschreibt. Hier ein Beispiel:

Dezimale Wertigkeit					
32	16	8	4	2	1
Dualzahl					
1	0	1	1	0	1

Das Ergebnis wäre dann

32+8+4+1 = 45

Die Verbindung zum hexadezimalen Zahlensystem ist recht einfach zu erledigen. Nehmen wir an, wir hätten diese Dualzahl:

1010 0101

Sie sehen schon, ich habe sie in Vierergruppen geteilt. Lege ich wieder die Wertigkeit darüber, sieht das so aus:

```
8 4 2 1   8 4 2 1
1 0 1 0   0 1 0 1
```

Das ergibt (von links) dezimal 10 und 5. In hex schreibt man für 10 aber A, also hieße die

```
A$ = "1"
WHILE A$ <> ""
  INPUT "Eine Zahl ($ wen hex)"; A$
  IF LEFT$(A$, 1) <> "$" THEN
    PRINT HEX$(VAL(A$))
  ELSE
    A$ = RIGHT$(A$, LEN(A$) - 1)
    L = LEN(A$)
    X% = 0
    FOR I=L TO 1 STEP -1
      X% = X% + VAL("&h" + MID$(A$,I,1)) * 16^(L-I)
    NEXT I
    PRINT X%
  END IF
WEND
```

Listing 1. Üben Sie mit diesem Programm in Amiga-Basic die Umwandlung von Dezimal- in Hex-Zahlen

Zahl in hex A5. Auch hier wieder mit Listing 2 ein Amiga-Basic-Programm zum Üben von Dualzahlen, auch Binärzahlen genannt. Wenn Sie eine Dezimalzahl eingeben, dann gibt sie das Programm in binär aus. Umgekehrt, geben Sie eine Binär-Zahl ein (erkenntlich am % als erstes Zeichen), erhalten Sie deren Wert in dezimal. Das Zeichen Prozent (%) ist der Präfix für Dualzahlen.

Der Stack (Stapel) hat eine enorme Bedeutung. Ohne ihn sind zum Beispiel keine Unterprogramme möglich. Der Stack ist ein Speicher (ein Stück RAM) mit besonderen Eigenschaften. Man nennt ihn auch LIFO für »Last In, First Out« oder Stapelspeicher. Sie packen Daten auf den Stack, indem Sie etwas auf den Stapel tun. Sie können (wie bei einem Stapel von Tellern) immer nur von oben (vom »Top of Stack«) etwas wegnehmen. Wenn Sie die Daten A, B und C in dieser Reihenfolge auf den Stack packen, können Sie sie nur in der Folge C, B, A zurücklesen.

Ganz wichtig: der Stack

Der Trick ist nun, daß die CPU tatsächlich niemals die Daten vom Stack nimmt, sondern nur die Daten woandershin kopiert. Gesteuert wird dieses durch den sogenannten Stapelzeiger, neudeutsch Stackpointer oder kurz SP. Der Stackpointer zeigt immer auf die Spitze des Stapels. SP ist also ein Zeiger, er sagt nicht, wie viele Daten auf dem Stack sind! Noch eine Vorbemerkung: Der Stack wächst von oben (den hohen Adressen) nach unten (zu den niedrigen Adressen hin). Die Anweisung »Packe A auf den Stack« bewirkt zwei Schritte:

1. Erniedrige SP,
2. Kopiere A in den Speicherbereich, auf den SP nun zeigt.

Die Umkehr, nämlich hole A vom Stack, hat zur Folge:

1. Kopiere Daten, auf die SP zeigt, nach A,
2. Erhöhe SP.

Die Daten auf dem Stack können auch Adressen sein. Ein GOSUB bewirkt nichts weiter, als daß die Adresse des dem GOSUB folgenden Befehls auf den Stack gepackt wird. Ein RETURN holt diese Adresse vom Stack und springt dann zum Befehl an dieser Adresse. Wird nun in einem Unterprogramm wieder GOSUB gesagt, sind in diesem Augenblick zwei Return-Adressen auf dem Stack. Ein RETURN führt dann in das Unterprogramm zurück, das nächste in das Hauptprogramm.

Daraus folgt auch, daß mit dem Stackmechanismus Unterprogramme beliebig tief geschachtelt werden können. Jedes Return erhöht den SP wieder und so hangelt man sich dann zurück. Aber Achtung, was passiert hier?

```
10 GOSUB 20
20 GOSUB 10
```

Da jedes GOSUB den SP erniedrigt, aber das Gegenstück, nämlich das RETURN fehlt, wächst der Stack nach unten. Er wird dann recht bald in Ihren Programm-Code laufen und den mit Return-Adressen überschreiben. Ergebnis: totaler Crash, auch in Basic. Probieren Sie es einmal. Die zweite Anwendung für den Stack ist die Parameterübergabe an Unterprogramme. Prinzipiell läuft das so: Es gibt in Assembler (nicht in Basic) die Befehle »Packe Daten auf den Stack« und »hole Daten vom Stack«. (Sie wissen, jeder Befehl impliziert ein Verändern des Stack-Pointers). Da kann ich dann sinngemäß schreiben:

```
10 A auf den Stack
20 B auf den Stack
30 GOSUB 100
   (Return-Adresse auf
   Stack).
```

und dann im Unterprogramm:

```
100 Hole Return-Adresse
    vom Stack (und
    merke sie)
110 Hole B vom Stack
120 Hole A vom Stack
130 Rechne mit A und B
140 Springe zur Return-
    Adresse
```

Was aber, wenn Ihr Unterprogramm mit Return enden soll? Dann schreibt man:

1. Return-Adresse auf den Stack,
2. Daten auf den Stack,
3. GOTO Unterprogramm.

Im Unterprogramm:

1. Daten vom Stack,
2. mit Daten arbeiten,
3. RETURN.

Wie schon gesagt: es gibt in Assembler keinen PRINT-Befehl, sondern nur die Möglichkeit, Bytes in einen Speicherbereich zu schreiben, der (vom Video-Controller) auf dem Bildschirm abgebildet wird. Überhaupt heißt Assemblerprogrammierung primär, Daten von einer Adresse auf eine andere Adresse zu bewegen. Auch die Peripherie-Geräte (Tastatur, Floppy u.s.w.) liegen beim Amiga innerhalb des Adreßbereichs (man nennt das »memory mapped«).

Die Geräte werden angesprochen, indem man bestimmte Daten in diese Adressen schreibt oder von Ihnen liest. Praktisch werden wir zwar die Hardware kaum so ansprechen, sondern die Parameter in Datenstrukturen eintragen und dann System-Routinen aufrufen, aber auch diese Datenstrukturen müssen wir adressieren.

Sie sehen also schon, die Adressierung als solche ist ganz wesentlich. Man kann eine Adresse auf sehr viele unterschiedliche Arten ansprechen, ein Beispiel hatten wir schon mit dem SP. Ich kann da sagen, stelle den SP auf die Adresse 4711. Ich kann aber auch sagen, hole Daten von der Adresse, auf die SP gerade zeigt (ohne zu wissen, wohin er zeigt).

diese Register durch spezielle Befehle unterstützt.

Natürlich entfällt auch der Transport vom Speicher zur CPU. Damit sind Registeroperationen wesentlich schneller als Zugriffe auf den Hauptspeicher und bieten (wegen der speziellen Befehle) einiges mehr an Komfort.

So gesehen ist eine CPU mit vielen Registern besser, als eine mit wenigen. Der 68000 hat viele Register, nämlich: 8 Datenregister, 7 Adreßregister, 2 Stackpointer, einen Programmzähler (PC) sowie ein Statusregister. Tabelle 1 zeigt den kompletten Registersatz des 68000.

Wie Sie sehen, sind die Register D0-D7 und A0-A7 plus PC je 32 Bit breit. Im Bild sind die Bits von 0 bis 31 numeriert. Das sind 4 Byte.

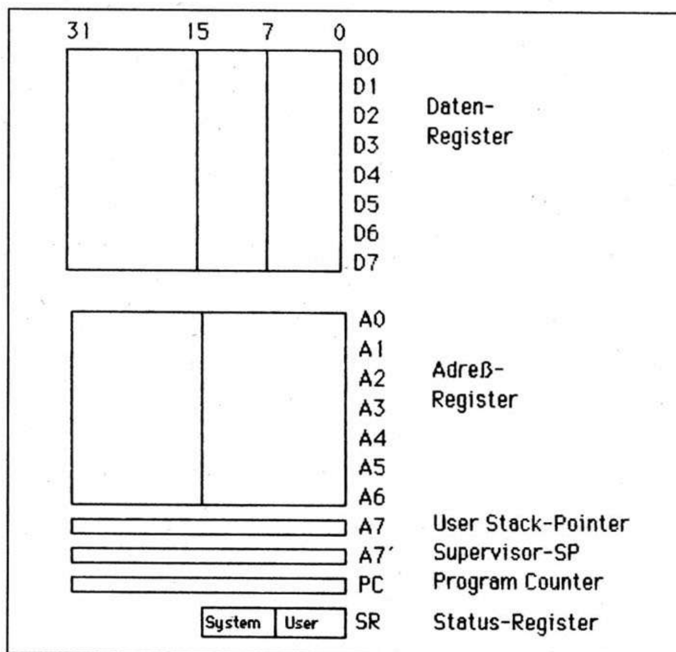


Tabelle 1. Alle Register des 68000-Prozessors

Das waren schon zwei Adressierungsarten. Insgesamt kennt der 68000 aber deren zwölf, und mit diesen zwölf Adressierungsarten werden wir uns gleich beschäftigen, doch zuerst müssen Sie die Register kennenlernen.

Bisher wissen wir, daß Daten im RAM oder ROM stehen. Daneben gibt es aber ein ganz spezielles RAM, der ein Teil der CPU ist. Dieser Speicherbereich besteht aus Gruppen von je 32 Bit und jede dieser Gruppen nennt man Register. Die Register werden nicht über Adressen, sondern über Namen angesprochen. Der Vorteil von Registern im Vergleich zum übrigen Speicher ist, daß sich die Register auf demselben Chip wie die CPU befinden, und die CPU den Zugriff auf

In einem Byte zählt man die Bits von 0 (niederwertiges Bit) bis 7 (höchstwertigtes Bit).

Zwei Bytes (16 Bit) nennt man Wort. Dessen Bits zählen von 0 bis 15. Zwei Worte (32 Bit) sind ein Langwort. Man spricht auch von den Datentypen Bit, Byte, Wort und Langwort. Die größten darstellbaren Zahlen in Abhängigkeit vom Typ zeigt diese Liste:

```
Bit      = 2↑ 1-1 = 1
Byte     = 2↑ 8-1 = 255
Wort     = 2↑16-1 = 65.535
Langwort = 2↑ 32-1 =
                4.294.967.299
```

In den Datenregistern kann man Bytes, Worte und Langworte ablegen.

Bei den Adreßregistern ist der Typ Byte nicht möglich. Die Stackpointer (A7) sind immer

»long«. Außerdem gibt es noch den Type BCD (Binary Coded Decimal). In diesem Falle wird ein Byte in zwei Halbbytes (nennt man Nibble) geteilt. Mit den 4 Bit eines Nibbles kann man nun die Zahlen 0 bis 15 darstellen. Auf 10 bis 15 wird dabei aber verzichtet, gültig sind in der BCD-Darstellung nur die Werte von 0 bis 9. Damit kann man in einem »BCD-Byte« immer zwei Zehner-Stellen darstellen. Ein Wort reicht also für eine vierstellige Dezimalzahl. Benötigt man mehr Stellen, muß man die entsprechende Anzahl Bytes sozusagen nebeneinanderlegen. Nun gibt es eine Unmenge kluger Algorithmen zum Thema BCD-Rechnerei. Für die armen Leute ohne 68000 sind die sehr wichtig. Wir können darauf verzichten, weil im 68000 die passenden Befehle dafür schon eingebaut sind.

Solange Sie nur mit Registern arbeiten, spielt die Größe (fast) keine Rolle, wenn Sie aber ein mit allen 32 Bit gefülltes Register in den RAM kopieren, belegt es da 4 Byte. Das ist ziemlich unpraktisch, weil Sie sehr oft mit Bytes oder Worten auskommen könnten, also auch mit weniger Speicher. Deshalb gibt es beim 6800 die Möglichkeit – genau: die Pflicht – bei jeder Operation, die Daten bewegt, anzugeben, welcher Typ dabei gilt.

Ein Beispiel: Der Datentransfer geschieht mit dem Befehl MOVE, und zwar mit der Syntax »MOVE Quelle,Ziel«. Tatsächlich bewegt der Befehl MOVE die Daten nicht, sondern kopiert sie, er kopiert von der Quelle auf das Ziel. Um die Daten im Register D3 auf die Adresse 4711 zu kopieren, schreibt man:

```
MOVE.B  D3,4711
```

oder

```
MOVE.W D3,4711
```

oder

```
MOVE.L D3,47711
```

Im Falle ».B« wird ein Byte kopiert, also die Bit 0-7 von D3, im Fall ».W« ist es ein Wort (Bit 0-15), im Fall ».L« das ganze Register. Im Hauptspeicher (hier ab Adresse 4711) werden dann entsprechend 1, 2 oder 4 Byte belegt. In welcher Reihenfolge dabei die verschiedenen Datentypen im RAM stehen, sollten Sie wissen, nämlich so, wie man sich das denkt.

Steht beispielsweise in D0 das Wort $AABB und wird D0 mit einem Move.W-Befehl auf die Adresse 1000 kopiert, so steht $AA in 1000 und $BB in 1001. Nur den Kollegen aus der 8-Bit-Ecke und denen, die vom IBM-PC kommen, sei noch einmal deutlich gesagt: Der 68000 speichert Daten in der richtigen Reihenfolge. Es werden nicht – wie bei den »8-Bitern/8088ern« – die Bytes eines Wortes vertauscht.

Wegen dieser freien Auswahl des Datentyps haben Sie auch die Pflicht, ihn bei den meisten Befehlen anzugeben. Fehlt der Typ, nehmen die meisten Assembler den Typ Wort an.

Um auf die Register zurückzukommen: Der Hauptunterschied zwischen Daten- und Adreßregistern ist, daß bei letzteren die Typen Bit und Byte nicht erlaubt sind.

Ansonsten können Sie durchaus auch Daten in Adreßregistern speichern und Adressen in Datenregistern. Letzteres werden wir beim Amiga häufig antreffen, da viele Routinen auch Adressen in Datenregistern erwarten.

Das Statusregister hat einen ganz besonderen Zweck. Damit wird in Assembler IF-THEN realisiert, ich komme später noch (sehr ausführlich) darauf zurück.

Wieviel Assembler-Befehle es gibt, ist gar nicht so einfach zu sagen. Je nach Zählweise können es 56 oder 1000 sein. Das liegt daran, daß ein Befehl je nach Adressierungsart (und weiteren Varianten) ganz unterschiedliche Wirkung zeigt. Beginnen wir mit dem Befehlsaufbau. Ein Befehl kann haben: keinen Operanden, einen oder zwei. Die Operanden können im Befehlswort selbst enthalten sein oder belegen bis zu vier weitere Worte, die dem Befehlswort unmittelbar folgen. Darüber brauchen Sie sich aber vorerst wenig Sorgen zu machen. Im Quelltext schreiben Sie den Befehl und die Operanden einfach hin, wieviel Worte das dann werden, ist Sache des Assemblers.

Adressierung ist alles

Ein Beispiel für einen Befehl ohne Operanden ist RTS (Return from Subroutine), was dem RETURN in Basic entspricht. Einen Operanden hätte der Befehl »CLR D0«. Das heißt: Lösche (Clear, fülle mit Nullbits) den Operanden D0 (das Register D0).

Ein Beispiel für einen Befehl mit zwei Operanden:

```
MOVE.L  A3,A4
```

Damit wird das Langwort im Register A3 nach A4 kopiert. Eine CPU ist um so besser, je mehr sinnvolle Adressierungsarten sie hat. Hier glänzt der 68000 ganz besonders. Dieser Luxus macht die Sache zwar auch etwas schwierig, denn das alles will gelernt sein. Dies stellt auch eine Barriere für die Kollegen dar, die gewohnt sind, mit den wenigen (und primitiven) Adressierungsarten der »8-Biter« auszukommen. Andererseits, wenn Sie dieses Thema beherrschen, dann beherrschen Sie auch den 68000.

Um zu zeigen, worum es geht: Im Beispiel von eben (MOVE.L A3,A4) wurde der Inhalt des Registers A3 nach A4 kopiert. Schreibe ich hingegen

```
MOVE.L   (A3),(A4)
```

heißt das, daß die Inhalte der Register als Adressen zu sehen sind. Hat beispielsweise im Moment des Befehls A3 den Wert 4711 und A4 ist gleich 5711, dann wird ein Langwort von Adresse 4711 (da startend und Byte für Byte) nach Adresse 5711 kopiert.

Wir haben nun schon zwei Adressierungsarten kennengelernt, nämlich »Register direkt« (MOVE.L A3,A4) und »Register indirekt« (MOVE.L (A3),(A4)). Um noch eine Stufe höherzugehen, schauen wir uns »Adreß-Register indirekt mit Postinkrement« an. Das sieht beispielsweise so aus:

```
MOVE.W (A0)+,D0
```

Im Klartext: Kopiere das Wort, auf das A0 zeigt, nach D0 und erhöhe danach (post) A0 um 2. Zwei deshalb, weil ein Wort 2 Byte hat. Bitte merken: Der 68000 ist eine Byte-Maschine, jede Adresse zeigt auf 1 Byte. »MOVE.L (A0)+,D0« würde ein Langwort kopieren und danach A0 um 4 inkrementieren. Die nächste Variante wäre »Adreßregister indirekt mit Predekrement. Ein Beispiel:

```
MOVE.L    D0,-(A5)
```

In diesem Falle wird vorab (pre) 4 (Langwort hat 4 Byte) von A5 subtrahiert, dann wird D0 dahin kopiert, wohin A5 nun zeigt.

Das hatten wir doch schon mal? Denken Sie an das Thema Stack. Daten werden auf den Stack gebracht, indem man den Stackpointer (SP) erniedrigt und dann die Daten auf die Adresse kopiert, auf die SP zeigt. Daten werden vom Stack geholt, indem man sie von der Adresse holt, auf die SP zeigt und dann SP erhöht. Das wäre dann unser schon bekanntes

```
MOVE.L (A5)+,D0
```

Tatsächlich kann man so jedes Adreßregister als Stackpointer einsetzen.

Die Besonderheit des Registers A7, das auch in vielen Assemblern SP heißt, liegt darin, daß dieses Register auch durch Befehle wie JSR (Jump to Subroutine) und RTS (Return) angesprochen wird. Man kann aber auch das mit anderen Registern erledigen, zum Beispiel kann man anstatt des Befehls RTS auch schreiben:

```
MOVE.L   (A7)+,A0
JMP      (A0)
```

Der MOVE-Befehl holt die Return-Adresse vom Stack in das Register A0, danach erfolgt ein Sprung (Jump) zur Adresse, auf die A0 nun zeigt. Sie sagen, warum der Umstand, ein RTS ist doch viel einfacher! Recht haben Sie, aber trotzdem werden Sie diese Lösung in Programmen sehen, die beispielsweise von Basic aus aufgerufen werden, und zwar so:

```
MOVE.L   (A7)+, 4711
...
```

(viele andere Befehle)

```
MOVE.L 4711,A0
JMP (A0)
```

Mit dem ersten MOVE-Befehl wird die Return-Adresse auf einen sicheren Platz in den RAM

```
a$ = "1"
WHILE a$ <> ""
  INPUT "Eine Zahl n (%n wenn binär) "; a$
  IF LEFT$(a$, 1) <> "%" THEN
     x% = VAL(a$)
     FOR i = 15 TO 0 STEP -1
       PRINT SGN(x% AND 2 ^ i);
     NEXT: PRINT
  ELSE
    a$ = RIGHT$(a$, LEN(a$) - 1)
    l = LEN(a$): x% = 0
    FOR i = l TO 1 STEP -1
    x% = x% + VAL("&h" + MID$(a$,i,1)) * 2^(l-i)
    NEXT
    PRINT x%
  END IF
WEND
```

Listing 2. Dual- in Dezimalzahlen wandelt dieses Basic-Programm

geholt (4711 ist hier nur symbolisch gemeint). Man sagt auch die Return-Adresse wird gerettet. Wenn irgend etwas schief geht, kann ich dann immer noch mit Hilfe dieser Adresse zu Basic zurück, egal wo der Stackpointer gerade steht.

Nach diesem Ausflug in die Praxis, der einmal andeuten sollte, wofür man verschiedenartige Adressierungsarten braucht, wieder zurück zur Theorie. Tabelle 2 zeigt eine Liste aller Adressierungsarten und noch etwas mehr.

Zuerst notieren Sie bitte nur, daß eine Adresse aus mehreren Angaben zusammengesetzt sein kann. Die CPU errechnet daraus die endgültige Adresse, auch effektive Adresse (ea) genannt. Wie Sie schon wissen, belegt das Befehlswort 16 Bit. Die vier höherwertigsten davon beschreiben den Befehl an sich. Die übrigen zwölf teilen sich in zwei Gruppen von je 6 Bit, die die Adressierungsart von Ziel und Quelle (so vorhanden) angeben. Die 6 Bit je Operand wiederum teilen sich in zwei Gruppen von je 3 Bit. Die eine Gruppe heißt Modus, die zweite Register. Mit 3 Bit sind die Zahlen 0 bis 7 darstellbar, deshalb gibt es auch die Register A0 bis A7 und D0 bis D7. Es gibt aber mehr als sieben Adressierungsarten. Das wird damit erreicht, daß nicht bei jedem Adressier-Modus alle Register erlaubt sind. Überhaupt sind bestimmte Adressierungsarten nicht für den Quell- und (gleichzeitig) den Zieloperanden erlaubt und außerdem auch nicht für jeden Befehl. Es ist wichtig, diese Einschränkungen genau zu kennen.

Wir hatten schon gesehen, daß in den einzelnen Bits eines Befehlswortes codiert ist, welche Daten dazugehören. Um das aufzuzeigen, wurde in Tabelle 2 der Modus und die Register-Nummer als Binärzahlen dargestellt. Steht dort »An« oder »Dn«, können Sie dafür %000 bis %111 (dezimal 0 bis 7) einsetzen.

Adressierungsarten im Detail

Wie Sie sicher erkannt haben, geht aus dem Befehlswort und den darin codierten Adressierungsarten auch hervor, wieviele Worte der Befehl im Speicher belegt. Register-Register-Adressierung (z.B. MOVE A0,A1) kommt mit einem Wort aus, geben Sie hingegen eine absolute Adresse an, kommt mindestens noch ein Wort hinzu. In den 16 Bit des Befehlswortes stecken alle Informationen, die die CPU benötigt, um den Befehl zu decodieren.

Ähnlich arbeiten auch sogenannte Disassembler. Das sind Programme, die aus dem Maschinencode wieder den Klartext der Assemblersprache bilden. Solange Sie aber ein solches Programm nicht schreiben wollen, können (und sollten) Ihnen diese Bitmuster herzlich egal sein. Es gibt viele Hacker, die den Hex-Code (die Maschinensprache) der »8-Bitter« lesen können, wie andere Leute die Zeitung. Diese Übung ist beim 68000 aussichtslos, also sehen wir die Sache von der praktischen Anwendung her, nämlich alle Adressierungsarten an je einem Beispiel.

Adressierungsarten des 68000

Adressierungsart	Kürzel	Modus	Register
Datenregister direkt	Dn	000	Dn
Adreßregister direkt	An	001	An
Adreßregister indirekt (ARI)	(An)	010	An
ARI mit Postinkrement	(An)+	011	An
ARI mit Predekrement	-(An)	100	An
ARI mit Adreßdistanz	d16(An)	101	An
wie vor plus Index	d8(An,Rg)	110	An
Absolut kurz	$XXXX	111	000
Absolut lang	$XXXXXXXX	111	001
PC-Relativ mit Adreß-Distanz	d16(PC)	111	010
wie vor plus Index	d8(PC,Rn)	111	011
Konstante, Statusregister	#,SR,CCR	111	100

Tabelle 2. Alle Adressierungsarten auf einen Blick

1. Register direkt

Eines der Register wird direkt angesprochen. Beispiel:

```
CLR  D0        (Lösche D0)
```

2. Adreßregister indirekt (ARI)

Der Inhalt des Registers ist eine Adresse, auf diese wirkt die Operation. Beispiel:

```
MOVE (A0),D0      ;ARI
```

(bitte merken!)

Das Wort, dessen Adresse in A0 steht, wird nach D0 kopiert.

3. ARI mit Postinkrement

Wirkt wie ARI, nur wird anschließend das Register inkrementiert. Beispiele:

```
MOVE.B      (A0)+,D0      ;
Kopie,dann A0=A0+1
MOVE.W      (A0)+,D0      ;
Kopie,dann A0=A0+2
MOVE.L      (A0)+,D0      ;
Kopie,dann A0=A0+4
```

4. ARI mit Predekrement

Wie ARI mit Postinkrement, nur wird das Register vor der Operation erniedrigt. Beispiel:

```
MOVE  -(A0),  D0          ;
A0=A0-2,  dann Kopie
```

5. ARI mit Adreßdistanz

Die effektive Adresse ist die Summe von Inhalt des Registers plus Adreßdistanz. Die Adreßdistanz ist eine vorzeichenbehaftete 16-Bit-Zahl im Bereich -32668...32767. Ein Beispiel:

```
MOVE -100(A0), D0
```

Wäre A0=500, würde das Wort von Adresse 400 nach D0 kopiert. Diese Adressierungsart sollten Sie sich gut merken! Beim Amiga werden wir diese sehr häufig brauchen.

6. ARI mit Adreßdistanz und Index

Nun wird es kompliziert. Die Adreßdistanz ist jetzt nur noch eine vorzeichenbehaftete 8-Bit-Zahl im Bereich von -128...127. Nun darf aber noch ein weiteres Register angegeben werden. Beispiel:

```
MOVE  100(A0,D0),4711
```

100 ist die Adreßdistanz, A0 enthält die Basisadresse, in D0 steht der Index. Alle drei werden addiert. Die Summe ist eine Adresse, das Wort (Byte, Langwort), das dort steht, wird ins Ziel (hier Adresse 4711) kopiert. Beim Index darf auch ein anderer Typ angegeben werden, also auch D0.B oder D0.L wären erlaubt, bei Adreßregistern als Index natürlich nur An.W und An.L (0<n<7). Auch der Index ist vorzeichenbehaftet, womit er im Falle Langwort im Bereich von 2 GByte liegen muß (wenn wir die mal hätten). Dieser Befehl ist ideal für die Abarbeitung von Tabellen und Arrays. Oft wird dabei die Adreßdistanz nicht benötigt (die Laufvariable steht im Indexregister), weshalb man zum Beispiel oft die Form »0(A3,D4.L)« sieht.

7. Absolute Adressierung

Dies ist der einfachste Fall. Beispiel:

```
MOVE  4711,5713
```

Das Wort von Adresse 4711 und 4712 wird auf Adresse 5713/14 kopiert. Die CPU unterscheidet dabei noch zwischen kurz und lang (Adreßbereich nur 64 KByte oder die vollen 16 MByte des 68000). Praktisch merken Sie den Unterschied kaum (die lange Adresse erfordert mehr Bytes in der Befehlslänge und ist etwas langsamer).

8. Konstanten-Adressierung

Auch wieder etwas ganz Einfaches. Um eine Konstante zu bewegen, brauchen Sie nur das Zeichen # vorzusetzen. Um beispielsweise das ASCII-Zeichen A in das Register D0 zu laden, schreiben Sie:

```
MOVE   #65,D0
```

oder

```
MOVE   #'A',D0
```

9. PC-relativ

Da muß ich etwas ausholen, damit Sie dieses Feature auch würdigen können. Sobald Sie in einem Assembler-Programm eine absolute Adresse angeben, ist das Programm an einen Ort im Speicher gebunden. Auch z.B »(A0)« (indirekt) ist in diesem Sinne absolut, denn vorher mußten Sie ja A0 mit einer Adresse versorgen.

Der von Ihnen vorgesehene Adreßbereich kann aber schon belegt sein, also muß Ihr Programm auch an einem anderen Ort laufen können. Dafür gibt es zwei Möglichkeiten. Erstens, das Programm ist verschiebbar (relokatibel). Dafür sorgt der Assembler, indem er mit dem Programm eine Tabelle aller absoluten Adressen speichert. Der Lader (oder das Programm selbst oder ein Utility) kann dann diese Adressen korrigieren, indem sie die Differenz zwischen geplanter und tatsächlicher Start-Adresse auf alle absoluten Adressen Tabelle addiert. Die zweite Möglichkeit ist, das Programm lageunabhängig (Position Independent) zu schreiben. In einem solchen Programm dürfen dann eben keine absoluten Adressen vorkommen, und genau da hilft der 68000 mit der PC-relativen Adressierung. Dabei wird die Adresse gerechnet als aktueller Stand des PC+Offset. Offset ist auch hier wieder auf -32768... 32767 begrenzt. Beispiel:

```
MOVE   100(PC),D0
```

10. PC-relativ mit Adreßdistanz und Index

Hier gilt sinngemäß das für »ARI mit Adreßdistanz und Index« Gesagte, nur daß die Basis-Adresse hier PC+2 ist. Beispiel:

```
MOVE   100(PC,A0.W),D0
```

Wie schon gesagt, ohne gute Kenntnisse des Betriebssystems ist Assemblerprogrammierung nicht möglich, also packen wir's an.

Das Betriebssystem des Amiga besteht grob vereinfacht aus drei Teilen, nämlich aus

- DOS,
- Intuition,
- Exec.

Die Aufgabe eines jeden OS (Operating System) ist es, die Verbindung des Computers mit der Außenwelt herzustellen. Dinge wie Zeichen von der Tastatur lesen, Zeichen auf dem Bildschirm darstellen oder Dateien von einer Diskette lesen sind typische Aufgaben des OS.

Nun kann der Amiga bekanntlich wie ein Standard-Computer bedient werden (Sie tun das, wenn Sie im CLI sind) oder über die grafische Benutzeroberfläche »Workbench«. Immer noch ganz grob (genauer behandeln wir das später) kann man nun sagen

- Standard = DOS,
- Grafik = Intuition.

Bliebe noch Exec, und das ist in unserem vereinfachten Modell primär für das Multitasking zuständig.

Das DOS (Disk Operating System) hat auch etwas mit Disketten zu tun, doch der Name untertreibt.

Tatsächlich kann das DOS auch mit der Tastatur und dem Bildschirm umgehen, ja sogar typische Amiga-Windows öffnen, den Drucker bedienen und einiges mehr. Weil das DOS recht einfach zu handhaben ist, werden wir uns vorerst nur damit beschäftigen. Schließlich müssen Sie zuerst die Assembler-Programmierung an sich lernen, was schon Stoff genug ist. Sie gleichzeitig noch mit den komplizierten Teilen der Amiga-System-Software zu behelligen, verkneife ich mir deshalb (aber nur vorerst).

Der Schlüssel zum Amiga

Aus Sicht des Programmierers ist das DOS eine Sammlung von Routinen (Unterprogrammen), die er alle benutzen darf. Jedes dieser Unterprogramme startet natürlich bei einer bestimmten Adresse, und demnach könnte man so ein Programm in der Form »JSR Adresse« aufrufen. Praktisch tut man das nicht, denn dann würde jede Änderung im OS dazu führen, daß sich einige oder alle dieser Adressen verschieben und somit alle »alten« Programme nur noch Makulatur wären. Deshalb gibt es eine Tabelle, in der die Startadressen aller Unterprogramme notiert sind.

Ändert sich ein UP und verschiebt damit sich (und andere) im Speicher, wird nur die Tabelle geändert. Wenn Sie also immer schön jedes Unterprogramm mittels dieser Tabelle aufrufen, werden Ihre Programme unter jeder Workbench und mit jedem Kickstart laufen, ansonsten garantiere ich für gar nichts.

Beim Amiga sind diese Unterprogramme nicht alle »auf einem Haufen« abgelegt, sondern in Gruppen sortiert, die man Libraries (Bibliotheken) nennt.

Eine Library ist vereinfacht ausgedrückt eine Sammlung von Unterprogrammen mit der zugehörigen Tabelle (je Unterprogramm ein Eintrag). Für jeden Zweck (zum Beispiel DOS, Intuition, Grafik) gibt es eine eigene Library.

Will man eine Funktion einer Library benutzen, muß man die Library mit »OpenLibrary« öffnen. Diese Funktion gibt einen Zeiger auf den Beginn (die Startadresse) der Tabelle zurück. Um nun ein Unterprogramm aufrufen zu können, muß man die Startadresse der Tabelle angeben und ein sogenanntes Offset, das die Differenz zwischen Startadresse und zugehörigem Tabellenplatz ist.

Verwaltet wird das Ganze vom sogenannten Library-Manager (ein Teil von Exec). Der Manager weiß, ob sich eine Library schon im ROM oder RAM befindet. Wenn nicht, versucht er, die Library von der Diskette zu laden.

Klappt das nicht (Library ist nicht auf der Diskette oder Speicher ist schon voll), gibt er 0 als Adresse zurück.

Der Umstand hat noch einen Grund: Wir haben einen Amiga und der unterscheidet sich von seinen Konkurrenten auch durch sein Multitasking-System. Das heißt vereinfacht (kommt auch noch genauer), daß gleichzeitig verschiedene Tasks (Programme) eine Library benutzen können.

Der erste Task wird die Library notfalls von der Diskette in das RAM laden (genau: das Laden veranlassen). Öffnen weitere Tasks dieselbe Library, wird der Manager nur noch die Adresse an diese Tasks melden.

Daraus folgt: Eine Library darf erst wieder aus dem Speicher gelöscht werden, wenn der letzte Task gemeldet hat, daß er sie ab sofort nicht mehr benötigt. Dafür gibt es die Funktion »CloseLibrary«. Jeder Task (also jedes Programm, das Sie schreiben) muß deshalb alle Libraries, die er geöffnet hat, auch wieder schließen. Andernfalls wird schnell der Speicherplatz knapp werden. Jedes Assemblerprogramm besteht aus den Feldern Marke, Befehle, Operanden (falls vorhanden) und Kommentar.

Hier ein Muster:

Der Kommentar muß nicht die nächste Zeile mit einem Befehl. In manchen Assemblern muß der Marke ein Doppelpunkt folgen; dann aber nur, wenn sie im Markenfeld steht, nicht wenn sie angesprochen wird.

Marke	Befehl	Operand(en)	Kommentar
Start	clr	d0;	Lösche Register
	move	d0,d1;	Befehl mit 2 Operanden
weiter			;nur Marke in der Zeile
	rts	kein Operand	

sein, trägt aber zur Lesbarkeit des Programms bei.

Je nach Assembler muß er mit einem Semikolon oder Stern beginnen, bei manchen Assemblern reicht auch der Platz (Kommentarfeld).

Steht der Kommentar allein in einer Zeile, muß »;« oder »*« sein.

Die Marke (Label) wird nur in einigen Fällen gebraucht. Sie kann auch allein in einer Zeile stehen, wirkt aber immer auf

Ausgenommen die Sonderfälle nur Marke oder nur Kommentar muß ein Befehl in einer Zeile stehen, und, so vorhanden, auch dessen Operand(en). Die Felder müssen durch mindestens eine Leerstelle voneinander getrennt sein. Meistens benutzt man die Tabulator-Taste (8ter Abstand). Nehmen Sie aber einen Texteditor, der dafür Blanks erzeugt.

Nun zu unserem ersten Programm (Listings 3a bis 3c). Nur

```
* A1_Met  Mein erstes Programm (Metacomco-Vers.)
* ---------------------------------------------

        INCLUDE "libraries/dos_lib.i"
        XREF    _DOSBase
        XREF    _SysBase
        XREF    _LVOOpenLibrary
        XREF    _LVOCloseLibrary
        XDEF    _main

_main   move.l  #dosname,a1             ;Name der DOS-Lib
        moveq   #0,d0                   ;Version egal
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOOpenLibrary(a6)     ;DOS-Lib oeffnen
        tst.l   d0                      ;Fehler?
        beq     fini                    ;wenn Fehler, dann Ende
        move.l  d0,_DOSBase             ;merke Zeiger

* Ausgabe-Handle ermitteln:

        move.l  _DOSBase,a6             ;DOS-Funktion rufen
        jsr     _LVOOutput(a6)          ;Hole Output-Handle
        move.l  d0,d4                   ;und in d4 merken

* nun Text ausgeben:

        move.l  d4,d1                   ;Ausgabe-Handle
        move.l  #string,d2              ;Adresse Text
        moveq   #20,d3                  ;Laenge Text
        move.l  _DOSBase,a6             ;Basis DOS
        jsr     _LVOWrite(a6)           ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!

        move.l  _DOSBase,a1             ;Basis d. Lib
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOCloseLibrary(a6)    ;Funktion "Schliessen"

fini    rts                             ;Return zum CLI

* Datenbereich:

dosname dc.b    'dos.library',0
        cnop    0,2
string  dc.b    'Hallo lieber Leser!',10
        cnop    0,2
        end
```

Listing 3a. Das erste Assembler-Programm (Metacomco-Version)

```
* A1_Seka   Mein erstes Programm (Seka-Version)
* ---------------------------------------------
SysBase:                    equ     4       ;Basis von Exec
LVOOpenLibrary:             equu    -552    ;Library oeffnen
LVOCloseLibrary:            equ     -414    ;Library schliessen
LVOOutput:                  equ     -60     ;DOS: Output-Handle holen
LVOWrite:                   equ     -48     ;Ausgabe

*DOS/Lib oeffnen:

main:   move.l  #dosname,a1             ;Name DOS-Lib
        moveq   #0,d0                   ;Version egal
        move.l  SysBase,a6              ;Basis Exec
        jsr     LVOOpenLibrary(a6)      ;DOS-Lib oeffnen
        tst.l   d0                      ;Fehler?
        beq     fini                    ;wenn Fehler, dann Ende
        move.l  d0,DOSBase              ;merke Zeiger

* Ausgabe-Handle ermitteln:

        move.l  DOSBase,a6              ;DOS-Funktion rufen
        jsr     LVOOutput(a6)           ;Hole Output-Handle und
        move.l  d0,d4                   ;in d4 merken
* nun Text ausgeben:
        move.l  d4,d1                   ;Ausgabe-Handle
        move.l  #string,d2              ;Adresse Text
        moveq   #20,d3                  ;Laenge Text
        move.l  DOSBase,a6              ;Basis DOS
        jsr     LVOWrite(a6)            ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!

        move.l  DOSBase,a1              ;Basis der Library
        move.l  SysBase,a6              ;Basis Exec
        jsr     LVOCloseLibrary(a6)     ;Funktion "Schliessen"

fini:   rts                             ;Return zum CLI
* Datenbereich:

DOSBase:        dc.l    0
                align 4
dosname: dc     'dos.library',0
                align 4
string: dc      'Hallo lieber Leser!',10
                align 4
```

Listing 3b. Das »Hallo«-Programm in der SEKA-Version

zur Einführung stelle ich Ihnen das Listing dreimal (für die Assembler Metacomco, SEKA und DevPac) vor. Alle weiteren Listings gelten für den DevPac-Assembler von HiSoft. Die Leser mit den anderen Assemblern sollten anhand ihrer Handbücher und der hier gegebenen Hinweise in der Lage sein, die Listings anzupassen. Beginnen wir mit Metacomco, der nach der klassischen Methode arbeitet.

Das Programm soll schlicht aber herzlich »Hallo lieber Leser« auf den Schirm schreiben und dann zum CLI zurückkehren. Vergessen wir vorerst den Kopf (Overhead) und betrachten das Listing ab der Zeile, die mit »_main« beginnt.

Auch für das simpelste Programm müssen wir eine Library öffnen. Dazu brauchen wir die Funktion OpenLibrary, die selbst in der Exec-Library zu finden ist. Wir brauchen aber auch immer zu jeder Funktion die Basis-Adresse der Library, und auch die erhält man mit OpenLibrary. Damit sich die Katze nun nicht in den Schwanz beißt, gibt es im Amiga eine feste Adresse (die einzige!), und das ist die Basis von Exec. Diese Adresse (4) hat den symbolischen Namen »_AbsExecBase« oder (hier benutzt) »_SysBase«. Der Schlüssel steckt in diesen Zeilen:

```
move.l  _SysBase,a6     ;
Basis Exec
jsr     _LVOOpenLibrary
(a6)    ;DOS-Lib oeffnen
```

Im Klartext: Lade das Register a6 mit der Konstanten »_SysBase«. Dann springe zum Unterprogramm (JSR = Jump to Sub Routine), dessen Adresse sich aus der Konstanten »_LVOOpenLibrary« und dem Register a6 berechnet.

Vorher müssen wir aber noch sagen, welche Library geöffnet werden soll. Dazu müssen zwei Parameter übergeben werden, nämlich der Name der Library und die Versionsnummer. Das erledigen die Zeilen

```
_main   move.l  #
dosname,a1      ;Name der
DOS-Lib
moveq   #0,d0           ;
Version egal
```

Die erste Zeile heißt: Kopiere (move) die Adresse von »dosname« in das Register a1. Das Doppelkreuz ist von immenser Bedeutung. Es heißt hier nämlich »Adresse von«. Vergessen Sie dieses Zeichen, gibt es einen bildschönen Absturz, denn dann heißt es »Inhalt von«. Bleibt noch die Versionsnummer. Es kann Libraries geben, die sich nicht im Namen aber in der Versionsnummer unterscheiden. Nur die Version 0 gibt es nie. Null ist reserviert für »nehme die erste (meistens die einzig vorhandene) Version«. Nach dem JSR kehrt das Unterprogramm zurück, und im Register d0 steht die Basisadresse der DOS-Library. Diese Adresse wird sofort in der Variablen »_DOSBase« gesichert.

Wir haben nun die Adresse der DOS-Library ermittelt und können damit arbeiten. Um einen Text ausgeben zu können, müssen wir zuerst wissen, wohin der Text geschrieben werden soll. Das kann eine Datei sein. Im Sinne von DOS ist aber auch das aktuelle Ausgabe-Gerät eine Datei mit dem speziellen Namen »Output«. Mit dem Start des Amiga (und solange wir nichts ändern) ist Output der Bildschirm (genau: das CLI-Fenster). Immer noch: Für DOS ist das eine Datei, ein File, und der Zugriff auf Files geschieht über sogenannte Handles. Normalerweise öffnet man einen File mit dem Befehl »Open«. Nur ist ja unser Output-File schon offen, und deshalb gibt es eine Funktion mit dem Na-

```
        opt     l-                              ;nicht linken!

* A1_Dev   Mein erstes Programm (DevPac-Version)
* ---------------------------------------------

_SysBase                equ     4       ;Basis von Exec
_LVOOpenLibrary         equ     -552    ;Library oeffnen
_LVOCloseLibrary        equ     -414    ;Library schliessen

_LVOOutput              equ     -60     ;DOS: Output-Handle holen
_LVOWrite               equ     -48     ;Ausgabe

*DOS/Lib oeffnen:

_main   move.l  #dosname,a1             ;Name der DOS-Library
        moveq   #0,d0                   ;Version egal
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOOpenLibrary(a6)     ;DOS-Lib oeffnen
        tst.l   d0                      ;Fehler?
        beq     fini                    ;wenn Fehler, Ende
        move.l  d0,_DOSBase             ;Zeiger merken

* Ausgabe-Handle ermitteln:

        move.l  _DOSBase,a6             ;DOS-Funktion rufen
        jsr     _LVOOutput(a6)          ;Hole Output-Handle
        move.l  d0,d4                   ;und in d4 merken

* nun Text ausgeben:

        move.l  d4,d1                   ;Ausgabe-Handle
        move.l  #string,d2              ;Address Text
        moveq   #20,d3                  ;Laenge Text
        move.l  _DOSBase,a6             ;Basis DOS
        jsr     _LVOWrite(a6)           ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!

        move.l  _DOSBase,a1             ;Basis der Lib
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOCloseLibrary(a6)    ;Funktion "Schliessen"

fini    rts                             ;Return zum CLI

* Datenbereich:

_DOSBase dc.l   0

dosname dc.b    'dos.library',0
        cnop    0,2

string  dc.b    'Hallo lieber Leser!',10
        cnop    0,2
```

Listing 3c. Die Version des ersten Programms für Devpac-Besitzer

men »_LVOOutput«, die die Handle von Output ermittelt. Genau das geschieht mit diesen Zeilen:

```
move.l  _DOSBase,a6        ;
DOS-Funktion rufen
jsr     _LVOOut
put(a6)  ;Hole Output-
Handle
move.l  d0,d4              ;
und in d4 merken
```

Wir rufen die DOS-Funktionen prinzipiell genauso auf, wie die Exec-Funktionen. Der Unterschied ist nur, daß jetzt das Register a6 auf die Basis der DOS-Library (_DOSBase) zeigt. Auch die Konstante »_LVOOutput« ist an anderer Stelle definiert (kommt noch). Wie alle Funktionen gibt auch »_LVOOutput« ihr Ergebnis in d0 zurück.

Da d0 ein Register ist, das auch andere Funktionen benutzen, retten wir es (kopieren es) in das Register d4. Jedenfalls haben wir nun die Handle im Register d4 und können damit arbeiten. Um auf einen File (oder ein Gerät) zu schreiben, braucht DOS diese Parameter:
- Handle in d1,
- Adresse, ab der die Daten zu finden sind in d2,
- Anzahl der Daten-Bytes in d3.

Schauen wir uns nun diese Zeilen an, so finden wir alles wieder:

```
move.l  d4,d1              ;
Ausgabe-Handle
move.l  #string,d2         ;
Addresse Text
moveq   #20,d3             ;
Laenge Text
```

So vorbereitet, können wir die Funktion »_LVOWrite« aufrufen:

```
move.l  _DOSBase,a6        ;
Basis DOS
jsr     _LVOWrite
(a6)    ;Funktion
"Schreiben"
```

Wie Sie sehen, ist das Prinzip immer gleich:

```
move.l  Basis_Adresse,a6
jsr     Offset(a6)
```

Nach dieser Methode wird dann auch zum Schluß des Programms die DOS-Library wieder geschlossen.

Im Datenbereich finden Sie nun einige Assembler-Direktiven. Wichtig ist die Direktive »dc.b«. Bitte beachten Sie: das ist eine Anweisung an den Assembler, kein 68000-Befehl.

»dc« heißt »define constant« (definiere Konstante), »dc.b« heißt dann Konstante vom Typ Byte. »string« ist ein Label, und die ganze Anweisung an den Assembler lautet nun: Setze ab (symbolischer) Adresse string die Zeichenfolge »Hallo...« ein.

»Hallo« wollen wir nun ausdrucken. Dazu benötigen wir einen Zeiger, der auf »Hallo« (genau: zuerst auf H) zeigt. Dazu ernennen wir das Register d2. Damit d2 mit der Adresse von string geladen wird, schreiben wir die Zeile

```
move.l  #string,d2         ;
Adresse Text
```

Nochmals: »dc.b« heißt definiere Konstante, und zwar hier vom Typ Byte. Im Operandenfeld stehen dann die Bytes. Diese können Sie einzeln eingeben (dc.b 100,33,20) als Text in Hochkommas oder wie hier gemischt. Der Name einer Library muß mit einem Null-Byte abgeschlossen sein, daher auch die Null am Ende des ersten »dc.b«.

Beim zweiten String ist kein Null-Byte erforderlich, weil die Write-Funktion die Länge als Parameter erwartet. Die 10 am Ende diese Textes ist der ASCII-Code für »neue Zeile«. Damit sorge ich dafür, daß nach dem Programmlauf der CLI-Prompt (1>) auf einer neuen Zeile startet.

»cnop 0,2« ist eine andere Form für »even«, die ich gewählt habe, weil sie zumindest der HiSoft- und der Metacomco-Assembler gleichermaßen verstehen (bei SEKA heißt das ALIGN). Sie erinnern sich: Man sollte Texte immer auf einer Wortgrenze beginnen lassen. Noch sicherer (und bei manchen Funktionen Pflicht) ist eine Langwortgrenze (cnop 0,4).

Nun bliebe noch die Frage zu klären, wo die Konstanten wie zum Beispiel »_LVOOpenLibrary« herkommen. Schauen wir uns dazu das SEKA-Listing (Listing 3b) an. Hier finden Sie in den ersten Zeilen sogenannte Equates. Auch das sind Assembler-Direktiven.

```
SysBase:   equ    4
```

definiert die Konstante SysBase mit dem Wert 4. Demnach sind diese Schreibweisen gleichwertig.

```
move.l   SysBase,a6
move.l   4,a6
```

Man sollte generell die erste Form wählen. Sowohl beim Metacomco- als auch DevPac-Assembler gibt es Include-Files (Textmodule), in denen diese Konstanten definiert sind.

Inzwischen dürfte Ihnen auch aufgefallen sein, daß sämtliche Offsets mit »_LVO« beginnen. LVO heißt Library Vector Offset. Bei SEKA gibt es da außer der DOS-Lib (und das ist nur 1/16 von allem) nichts. Die Unterschiede von SEKA zum Standard zeigt man am besten anhand der DevPac-Lösung (Listing 3c).

Bei Metacomco und DevPac beginnen alle Konstanten mit dem Unterstrich. Dieser ist bei SEKA als erstes Zeichen nicht erlaubt. »dc.b« kennt SEKA nicht, man muß das b weglassen.

Für DOS_Base habe ich mit »dc.l 0« diese Adresse mit dem Langwort 0 belegt. An sich brauche ich aber nur Speicherplatz für diese Variable, wofür man normalerweise »ds.l 1« schreibt (definiere Speicher für 1 Langwort). Da SEKA diese Direktive nicht kennt, bin ich auf »dc.l« ausgewichen.

In der Metacomco-Lösung (Listing 3a) fehlt das ganz. Dafür sehen Sie am Anfang des Listings sehr oft XDEF. Das heißt diese Dinge sind extern definiert, genau: im File »amiga. lib«. Bei Metacomco ist nach dem Assemblieren noch ein Linker-Lauf erforderlich. Dabei werden auch die externen Referenzen behandelt. SEKA und DevPac kommen ohne Linkerlauf aus, beziehungsweise können mit dem Assembler schon lauffähige Programme erzeugen.

Wenn Sie nun dieses Programm mit einem Editor eingetippt haben, geht die Arbeit erst los. Speichern Sie den Text zum Beispiel unter dem Namen TEST.S und kehren zum CLI zurück. Nun folgen mehrere Schritte, die von Ihrem Assembler-Paket abhängen. Lesen Sie bitte in Ihrem Handbuch nach.

Bei SEKA und HiSoft ist die Sache ganz einfach. Geben Sie das A-Kommando beziehungsweise <Amiga A> bei Hisoft. Bei letzterem sollten Sie noch darauf achten, daß »nicht linkbarer Code« gewählt wurde, womit ein sofort ausführbares Programm entsteht.

Dazu können Sie im Listing »opt l-« als erste Zeile eingeben. Bitte beachten Sie: es muß wirklich die erste Zeile sein, auch eine Leerzeile davor ist nicht erlaubt.

Auch bei Metacomco müssen Sie zuerst assemblieren. Hier ist dann noch ein Linker-Lauf erforderlich. Da beides viel Tipperei bedeutet, schreiben Sie am besten das Batch-File aus Listing 4 und speichern es unter dem Namen »make« im s-Directory. Tippen Sie nun

```
execute make test
(ohne .s!)
```

Nun mögen ja Texte, die wir ausgeben, ganz informativ sein, aber im allgemeinen erwarten wir wohl auch Eingaben von der Tastatur. Mit Listing 5 gehen wir deshalb einen klei-

```
.key file/a
c/assem <file>.s -o <file>.o -c s -i include
c/alink <file>.o to <file> library lib/amiga.lib
```

Listing 4. Das Batchfile für die Metacomco-Variante von Listing 3

```
        opt     l-                              ;nicht linken!

* Stringeingabe

_SysBase                equ     4       ;Basis v. Exec
_LVOOpenLibrary         equ     -552    ;Library oeffnen
_LVOCloseLibrary        equ     -414    ;Library schliessen

_LVOOutput              equ     -60     ;DOS: Output-Handle holen
_LVOWrite               equ     -48     ;Ausgabe
_LVORead                equ     -42
_LVOInput               equ     -54

*DOS/Lib oeffnen:

_main   move.l  #dosname,a1             ;Name der DOS-Library
        moveq   #0,d0                   ;Version egal
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOOpenLibrary(a6)     ;DOS-Lib oeffnen
        tst.l   d0                      ;Fehler?
        beq     fini                    ;Ende wenn Fehler
        move.l  d0,a6                   ;merke Zeiger

* Ausgabe-Handle ermitteln:

        jsr     _LVOOutput(a6)          ;Hole Output-Handle
        move.l  d0,d4                   ;und in d4 merken
        move.l  d4,d1                   ;nun Text ausgeben
```

Listing 5. Eingabe von Strings in Assembler

```
          move.l  #string,d2          ;wie gehabt
          moveq   #12,d3
          jsr     _LVOWrite(a6)

* Nun lese von der Tastatur

          jsr     _LVOInput(a6)       ;Hole Input-Handle
          move.l  d0,d1               ;und in d1 kopieren
          move.l  #buffer,d2          ;Adresse des Puffers
          moveq   #80,d3              ;erlaube 80 Zeichen
          jsr     _LVORead(a6)        ;und lese
          move.l  d0,len

* nun Inhalt buffer ausgeben:

          move.l  d4,d1               ;Ausgabe-Handle
          move.l  #buffer,d2          ;Address Text
          move.l  len,d3              ;Laenge Text
          jsr     _LVOWrite(a6)       ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!
          move.l  a6,a1               ;DOS-Lib-Basis
          move.l  _SysBase,a6         ;Basis Exec
          jsr     _LVOCloseLibrary(a6);Funktion "Schliessen"

fini      rts                         ;Return zum CLI

* Datenbereich:

dosname dc.b    'dos.library',0
          cnop    0,2

string  dc.b    'Enter Text: '
          cnop    0,2
buffer  ds.b    80
len       ds.l    1
```

Listing 5. (Schluß)

nen Schritt weiter. Bis hin zur Zeile »* nun lese von der Tastatur« hat sich gegenüber dem ersten Listing nichts geändert. Der ausgegebene Text heißt jetzt »Enter Text: «, und an dieser Stelle soll der User lostippen. Damit wir wissen, woher die Eingaben kommen, müssen wir zuerst die Handle der Tastatur kennen. Analog zur Output-Handle ermitteln wir diese mit der Funktion »_LVOInput«. Haben wir die Input-Handle, können wir damit »_LVORead« aufrufen.

Diese Funktion verhält sich sonst wie »_LVOWrite«, nur daß wir hier nicht die Adresse eines Textes übergeben, sondern die Adresse eines Puffers, in dem die Eingabe abgelegt werden soll. Mit

```
buffer  ds.b    80
```

weist man den Assembler an, nun eine Lücke von 80 Byte zu erzeugen. Mit »ds.w 40« oder »ds.l 20« hätte ich dasselbe erreicht. Auf die gleiche Art wird ein Langwort reserviert, in dem wir die Länge der Eingabe ablegen können. Die Funktion »_LVORead« ist sehr flexibel. Die Längenangabe, mit der sie aufgerufen wird, ist immer das Maximum. Geben Sie weniger ein (Ende mit Return-Taste), steht nach dem JSR die Ist-Länge im Register d0. Ich kopiere dann d0 in die Variable »len«. Das muß hier zwar nicht sein, sollte aber einmal gezeigt werden. Anschließend wird nämlich wieder in das Register d3 kopiert, um die Ihnen schon bekannte Funktion »_LVO Write« aufzurufen, die jetzt allerdings den Pufferinhalt ausgibt.

In diesem Abschnitt geht es um Schleifen. Vorab möchte ich allerdings etwas zur Rationalisierung unserer Arbeit tun. Wir haben nun schon zweimal die Zeilen von Listing 6 eingetippt. Speichern Sie den Text aus Listing 6 mittels der Block-Funktion Ihres Editors unter dem Namen »OpenDos.i«. In den folgenden Programmen reicht dann eine Include-Anweisung, und der Assembler wird diesen Text automatisch einziehen.

Bei Metacomco muß der Datei-Name in Anführungszeichen oder Hochkommas stehen. SEKA kennt kein Include, dort müssen Sie den Text mittels des R-Kommandos lesen.

Die folgende Aufgabe lautet: Es sollen die Buchstaben von A bis Z in einen Puffer geschrieben werden, dann soll der ganze Puffer mittels einer DOS-Funktion ausgedruckt werden. Eine Lösung wäre diese Methode:

```
lea.l   buffer,a0
move.b  #'A',(a0)+
move.b  #'B',(a0)+
move.b  #'C',(a0)+
...
```

Daß das nicht besonders intelligent ist, dürfte klar sein. Aber auch dazu müssen die Zeichen in einem Puffer stehen, die Frage ist nur, wie wir sie da hineinbekommen. Nun, man stellt einen Zeiger, hier ein Register, auf den Beginn des Puffers, und zwar mit

```
lea.l   buffer,a0
```

Das heißt »Lade a0 mit der effektiven Adresse von buffer«. Das ».l« ist an sich überflüssig (Adressen sind immer lang), aber manche Assembler bestehen trotzdem darauf. Der Befehl ist von gleicher Wirkung wie

```
move.l  #buffer,a0
```

Mit der Anweisung

```
move.b  #'A',(a0)+
```

wird die Konstante A auf die Adresse geschrieben, auf die a0 zeigt (hier Beginn buffer) und anschließend wird a0 um eins hochgezählt. Ein beliebter Fehler ist übrigens, das ».b« wegzulassen. Die Assembler setzen dafür nämlich automatisch ».w« ein, womit in diesem Fall a0 um 2 erhöht wird. a0 zeigt schon auf die nächste Adresse, und wir könnten somit das Spielchen fortsetzen. Genau das wollen wir aber nicht, sondern anstatt der vielen Einzelschritte eine Schleife einsetzen. Listing 7 bringt die Lösung.

Im Gegensatz zu einigen Konkurrenten hat der 68000 einen Schleifenbefehl eingebaut, nämlich:

```
DBcc    Dn,Marke
```

Das steht für »Decrement and Branch on Condition Code«. Wow, das ist ein Ding!

Also der Reihe nach: Mit dem DBcc-Befehl wird immer ein Datenregister angegeben, das kann D0 bis D7 sein, nennen wir es Dn. Vor dem Eintritt in die Schleife wird Dn ein Wert zugewiesen. In der Schleife, genau immer dann, wenn der DBcc-Befehl durchlaufen wird, wird Dn um eins dekrementiert. Solange Dn dabei nicht -1 wird, erfolgt ein Sprung zu »Marke«, ansonsten wird der nächstfolgende Befehl ausgeführt. Nun zum »cc«: Zusätzlich kann man nun noch vor dem DBcc-Befehl mit zum Beispiel einer CMP-Anweisung (compare = vergleiche) eine Bedingung testen und dann zum Beispiel sagen:

```
CMP     (A0)+,D0
DBeq    D1,Marke
```

In diesem Falle erfolgt der Sprung zu »Marke« nur dann, wenn die Bedingung »A(0) eq (equal = gleich) D0« erfüllt ist, ansonsten wird die Schleife beendet. Man kann es auch so sehen: Die Schleife wird durchlaufen, solange die cc-Bedingung nicht erfüllt ist, aber nur so lange, bis der Zähler auf -1 ist. Die Kürzel für »cc« sind übrigens die gleichen, wie beim bcc-Sprung-Befehl. Zum Beispiel gibt es BEQ (Branch if Equal) und DBEQ (Decrement and Branch if Equal). Die Einzelheiten zu allen »CCs« kommen noch, machen wir aber erst einmal mit der Praxis weiter. Häufig interessiert nämlich die Bedingung überhaupt nicht, man will nur zählen. In diesem Fall sagt man einfach:

```
DBRA
```

Das bedeutet »Decrement and Branch Allways« (springe immer), natürlich nur solange der Zähler nicht abgelaufen ist. Häufig sieht man auch »DBF«, wobei F für »False« (Falsch) steht, das ist nur eine andere Schreibweise. Gute Assembler akzeptieren sowohl »RA« als auch »F«. Nun können wir uns dem Listing 7 zuwenden. Wir

```
_SysBase             equ     4        ;Basis von Exec
_LVOOpenLibrary      equ     -552     ;Library oeffnen
_LVOCloseLibrary     equ     -414     ;Library schliessen

_LVOOutput           equ     -60      ;DOS: Output-Handle holen
_LVOWrite            equ     -48      ;Ausgabe
_LVORead             equ     -42
_LVOInput            equ     -54

*DOS/Lib oeffnen:

_main   move.l  #dosname,a1           ;Name der DOSS-Lib
        moveq   #0,d0                 ;Version egal
        move.l  _SysBase,a6           ;Basis Exec
        jsr     _LVOOpenLibrary(a6)   ;DOS-Lib oeffnen
        tst.l   d0                    ;Fehler?
        beq     fini                  ;wenn Fehler, Ende
        move.l  d0,a6                 ;Zeiger merken
```

Listing 6. Dieses Include-File wird noch häufig benötigt

```
        opt     1-                          ;nicht linken!
* Drucke A-Z
        include OpenDos.i
* Ausgabe-Handle ermitteln:
        jsr     _LVOOutput(a6)              ;Hole Output-Handle
        move.l  d0,d4                       ;und in d4 merken

* Puffer fuellen

        lea.l   buffer,a0
        move    #25,d0                      ;Siehe Text
        move.b  #'A',d1

loop    move.b  d1,(a0)+
        addq    #1,d1
        dbra    d0,loop
        move.b  #10,(a0)                    ;Neue Zeile

* nun Inhalt buffer ausgeben:
        move.l  d4,d1                       ;Ausgabe-Handle
        move.l  #buffer,d2                  ;Address Text
        move.l  #27,d3                      ;Laenge Text
        jsr     _LVOWrite(a6)               ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!

        move.l  a6,a1                       ;DOS-Lib-Basis
        move.l  _SysBase,a6                 ;Basis Exec
        jsr     _LVOCloseLibrary(a6)        ;Funktion "Schliessen"

fini    rts                                 ;Return zum CLI

* Datenbereich:

dosname dc.b    'dos.library',0
                cnop    0,2

buffer  ds.b    80
```

Listing 7. Drucken von A bis Z – mit DBcc

```
        opt     1-                          ;nicht linken!

*Kommandozeile ausgeben

* Immer zuerst Adresse und Laenge der Kommandozeile retten
        movem.l a0/d0,-(sp)

        include OpenDos.i

        jsr     _LVOOutput(a6)              ;Hole Output-Handle
        move.l  d0,d1                       ;da soll sie hin

        movem.l (sp)+,a0/d0                 ;Parameter zurueck

        move.l  a0,d2                       ;Adresse Kommandozeile
        move.l  d0,d3                       ;Laenge
        jsr     _LVOWrite(a6)               ;Funktion "Schreiben"

* Zum Schluss immer die Lib schliessen!
        move.l  a6,a1                       ;DOS-Lib-Basis
        move.l  _SysBase,a6                 ;Basis Exec
        jsr     _LVOCloseLibrary(a6)        ;Funktion "Schliessen"

fini    rts                                 ;Return zum CLI

* Datenbereich:

dosname dc.b    'dos.library',0
```

Listing 8. Dieses Programm liest die Kommandozeile

wollten ja die 26 Buchstaben von A bis Z drucken.

Weil der Zähler d0 aber immer bis -1 läuft, initialisiere ich ihn mit 25, siehe erste Zeile. Den Code für Buchstaben halte ich im Register d1, das wird also zuerst mit »A« geladen.

Bei »Loop« geht es nun los. Wie gehabt, packen wir ein Zeichen mittels »(a0)+« in den Puffer. Doch nun das Neue: Mit »addq #1,d1« wird d1 inkrementiert, aus dem A (ASCII-Code 65) wird also ein B (66) (dann aus dem B ein C usw.). Die Arbeit leistet die nächste Zeile.

```
dbra    d0,loop
```

Dekrementiere d0, wenn es dann noch nicht -1 ist, springe zu »Loop«, ansonsten nächster Befehl. Hier ginge es also im Falle von -1 bei der Ausgabe weiter.

```
...
MOVE.L 4711,A0
JMP (A0)
```

Um wieder auf das DOS zu kommen, möchte ich Ihnen noch ein Programm vorstellen, das einen Text ausgibt. Diesmal aber einen Text, der im Programm nirgends definiert ist. Sie rufen bekanntlich ein Programm unter CLI auf, indem Sie nur seinen Namen eintippen. Sie können aber dem Namen nach einem Leerzeichen noch beliebigen Text folgen lassen. Diesen Text nennt man die Kommando-Zeile. Viele CLI-Kommandos arbeiten damit. Tippen Sie zum Beispiel

```
cd df0:
```

so rufen Sie damit das Programm »cd« auf und übergeben die Kommandozeile »df0:« als Parameter. Wie man das Kommando liest, zeigt Listing 8. Das DOS speichert die Kommandozeile im RAM und stellt das Register a0 auf die Anfangsadresse. In d0 wird die Länge notiert. Da prinzipiell die Register d0, d1, a0 und a1 »Scratch«, also Schmierpapier sind (jede Routine kann sie ändern), muß ein Programm, das die Kommandozeile benötigt, zuerst die Register a0 und d0 retten. Normalerweise speichert man beide Register in Variablen ab, ich möchte Ihnen aber eine andere Möglichkeit vorstellen, und das wäre der Stack.

Der 68000 kann mit einem einzigen Befehl alle oder einige Register auf den Stack bringen oder von dort holen. Mit

```
movem.l  a0/d0,-(sp)
```

werden a0 und d0 auf dem Stack abgelegt. Es sind beliebige Kombinationen wie »a0/a3/a5/d1/d6« erlaubt, oder Listen wie »d0-d7/a0-a4« (d0 bis d7, a0 bis a4). Wichtig ist, daß Sie später mit dem analogen Befehl die Daten auch wieder vom Stack holen. In unserem Fall geschieht das mit

```
movem.l (sp)+,a0/d0      ;
Parameter zurueck
```

Danach müssen wir nur noch die Register in die Parameter-Register der Write-Funktion kopieren und können die Kommandozeile ausgeben.

Das nächste Programm soll fragen »Wie heißt Du?« Der User gibt dann einen Text ein (ich hoffe, seinen Namen), und das Programm antwortet »Guten Tag, lieber ....«. Alles, was wir dafür brauchen, kennen Sie schon (Eingabe und Ausgabe von Strings), nur langsam wird die Sache lästig. Wir müssen nämlich dreimal einen Text ausgeben. Jedesmal die ganze Sequenz dafür zu schreiben, ist zwar möglich, aber besser erledigt man das mit Unterprogrammen. Das Problem bei Unterprogrammen ist die Parameterübergabe. Wenn ich alle drei Parameter der Write-Funktion übergebe (drei Befehle) und dann mein Unterprogramm »print« aufrufe, habe ich nichts gewonnen. Ich könnte dann auch gleich die drei Parameter an Write übergeben und diese DOS-Routine rufen. Daher folgende Vereinbarungen:

1. Die Output-Handle ist bekannt (steht in d4)
2. Es wird nur die Adresse des Textes übergeben. Im Text ist die Länge »versteckt«. Wie man das löst, zeigt Listing 9.

Hauptsache Unterprogramme

Der Trick steckt in den letzten Zeilen. Die ersten Bytes der Strings msg1 und msg2 halten die Länge des folgenden Textes. Da Strings in Pascal so abgelegt werden, spricht man auch von Pascal-Strings.

Der Aufruf des Unterprogramms erfolgt nach diesem Schema:

```
lea.l   msg1,a0 ;frage
nach Namen
bsr             print
```

Das heißt, nur die Adresse des jeweiligen Strings wird übergeben. BSR heißt »Branch to Sub Routine« (Verzweige zum Unterprogramm). Der Unterschied zu JSR besteht darin, daß BSR auf eine Sprungweite von ±32 KByte begrenzt ist, während JSR für den vollen Adreßbereich des 68000 (16 MByte) gilt. Sie müssen nicht BSR nehmen, das spart nur etwas Code und Zeit.

Nun zum Unterprogramm selbst (ab Label »print« im Listing). In der zweiten Zeile

```
(move.b (a0)+,d3
```

steckt des Pudels Kern: das Längen-Byte wird in das Register d3 kopiert (da erwartet es Write). Das kleine »+« stellt gleichzeitig a0 auf den Beginn des Textes. Wir können dann getrost zwei Zeilen später a0 in d2 kopieren. Das klappt, nur der direkte Zugriff auf den Textbeginn geht schief (ungerade

```
        opt     l-                      ;nicht linken!

* Unterprogramm-Demo

        include OpenDos.i

        jsr     _LVOOutput(a6)          ;Hole Output-Handle
        move.l  d0,d4
        lea.l   msg1,a0                 ;frage nach Namen
        bsr     print

        jsr     _LVOInput(a6)           ;Hole Input-Handle
        move.l  d0,d1                   ;und arbeite damit
        lea.l   buffer,a2               ;Zeiger auf Puffer
        move.l  a2,d2                   ;an Read uebergeben
        addq.l  #1,d2                   ;Laengen-Byte skippen
        move.l  #79,d3                  ;solang darf Name sein
        jsr     _LVORead(a6)            ;Ist-Laenge in d0
        addq.l  #1,d0                   ;verlaengern
        move.b  d0,(a2)                 ;und eintragen
        move.b  #'!',-1(a2,d0.l)        ;! in Puffer
        move.b  #10,0(a2,d0.l)          ;und noch neue Zeile

        lea.l   msg2,a0                 ;sage Guten Tag
        bsr     print

        move.l  a2,a0                   ;drucke Namen
        bsr     print

* Zum Schluss immer die Lib schliessen!

        move.l  a6,a1                   ;DOS-Lib-Basis
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOCloseLibrary(a6)    ;Funktion "Schliessen"

fini            rts                     ;Return zum CLI

print   clr.l   d3
                move.b(a0)+,d3          ;Laenge
                move.l  d4,d1           ;Output-Handle
                move.l  a0,d2           ;Adresse
                jsr     _LVOWrite(a6)   ;Funktion "Schreiben"
                rts

* Datenbereich:

dosname dc.b    'dos.library',0
                cnop    0,2

msg1            dc.b    16,'Wie heisst Du? '
                cnop    0,2

msg2            dc.b    19,10,'Guten Tag, lieber '
                cnop    0,2

buffer          ds.b    80
```

Listing 9. Unterprogramme im praktischen Einsatz

Adresse, Write muß diesen Fall abfangen).

Nun zur ersten Zeile: Die Länge muß als Langwort übergeben werden, wir haben aber nur ein Byte. Unser Problem: Stehen in einem Register die 4 Byte »B3 B2 B1 B0«, dann »moved«

```
move.b:               B0
move oder move.w: B1, B0
 move.l:  B3, B2, B1, B0
```

Wenn wir nur 1 Byte übertragen, bleibt der Rest von 3 unverändert. So können dann im Langwort d3 »Hausnummern« entstehen. Daher lösche ich vorher mit clr (clear = lösche, fülle mit Nullen) das Register.

Hat man sich einmal auf ein bestimmtes Verfahren zur Parameterübergabe an Unterprogramme eingelassen, dann kann das unter Umständen gravierende Folgen haben. Um das zu zeigen, habe ich mir die Auflage erteilt, daß auch Texte, die mit Read gelesen wurden, mit der Print-Routine ausgegeben werden sollen.

Read liest bekanntlich in einen Puffer, und zwar an sich ab dessen Beginn. Nun erwartet aber Print als erstes Zeichen in diesem Puffer die Länge des Textes. Das hat zur Folge:

```
lea.l   buffer,a2      ; Zeiger auf Puffer
move.l  a2,d2          ; an Read uebergeben
addq.l  #1,d2          ; Laengen-Byte skippen
move.l  #79,d3         ; solang darf Name sein
jsr     _LVORead(a6)   ;Ist-Laenge in d0
move.b  d0,(a2)        ; und eintragen
```

Die ersten beiden Zeilen sind noch die üblichen. Wir stellen a2 als Zeiger auf den Puffer-Beginn und kopieren dann a2 nach d2, wo Read üblicherweise die Pufferadresse erwartet. Doch nun wird d2 um 1 erhöht. Damit zeigt d2 auf das zweite Byte im Puffer, Read wird also ab dieser Adresse den Puffer füllen, unser Längen-Byte bleibt frei. Nach dem JSR wird dann einfach die Ist-Länge in den Puffer kopiert, was

```
move.b  d0,(a2)
```

erledigt. Doch nun schauen Sie auf das Listing, da steht noch mehr. Der Grund: Nach dem Namen soll noch ein Ausrufungszeichen gedruckt werden. Dafür wird

a) die Ist-Länge erhöht und
b) das Ausrufungszeichen in den Puffer geschrieben. Dahinter soll dann noch wieder die 10 (neue Zeile) folgen. Daher:

```
move.b  #'!',-1(a2,d0.l)     ;! in Puffer
move.b  #10,0(a2,d0.l) ; und noch neue Zeile
```

Falls Sie sich bisher noch nicht vorstellen konnten, was man mit »ARI mit Index und Offset« (siehe Adressierungsarten) anfangen könnte, hier hätten wir eine Anwendung.

Zuerst stolpern Sie vielleicht über das »-1«. Dazu muß man wissen, daß die Read-Funktion auch die Return-Taste (ASCII-Code 10) als letztes Zeichen im Puffer ablegt und auch bei der Länge mitzählt.

```
(a2, d0.l)
```

heißt: bilde die Adresse aus Summe von a2+d0. Beginnt der Puffer zum Beispiel auf Adresse 1000 und haben wir die Zeichen ABC eingetippt, so stehen im Puffer

Adresse	=	1000	1001	1002	1003
Zeichen	=	A	B	C	Return

Die Länge ist 4. Folglich ist

```
1000+4 (a2 + d0) = 1004
```

Wir wollen aber das Aufrufungszeichen auf die Adresse 1003 bringen. Daher addieren wir noch das Offset von -1, sprich subtrahieren 1. Der nächste Befehl

```
move.b  #10,0(a2,d0.l)
```

addiert ein Offset von null. Damit wird dann »#10« auf die Adresse 1004 geschrieben. Wir können die Null nicht in jedem Assembler weglassen. Die Syntax des Befehls will dort eine Konstante sehen. Übrigens findet man häufig diese Form mit dem Null-Offset, weil es meistens reicht, die Adresse nur aus den beiden Registern zu bilden.

Schauen wir uns die Sequenz »IF Bedingung THEN GOTO« einmal genauer an. Prinzipiell funktioniert es wie in den Hochsprachen: man fragt eine Bedingung ab und verzweigt in Abhängigkeit vom Ergebnis. Der kleine Unterschied zu den Hochsprachen: Es sind zwei Befehle (Abfrage und Sprung), und die Bedingung ist der Zustand einiger Bits im CCR (Condition Code Register). Diese Bits - auch Flags genannt - haben Namen, und zwar heißen sie X, N Z, V und C.

IF THEN und die Flags

Es gibt viele Befehle, die diese Flags beeinflussen, meistens jedoch sind es mathematische Operationen mit zwei Operanden, wobei der Quelloperand vom Zieloperanden subtrahiert wird. Wird dabei das Ergebnis negativ, setzt der 68000 das N-Flag (Bit wird 1).

Entsteht ein Überlauf, wird das Overflow-Bit (V) gesetzt, bei einem Übertrag (Addition) oder beim »Borgen« während der Subtraktion geht das Carry-Flag (C) auf Eins. Umgekehrt, treten diese Zustände während der Operation nicht ein, werden die entsprechenden Flags auf Null gesetzt.

Die Flags selbst kann man zwar abfragen, aber das ist nicht üblich. Statt dessen schreibt man einen Befehl, der die Flags beeinflußt (prüft) und dann sinngemäß »GOTO Adresse, wenn dieses Flag diesen Zustand hat«. Merken Sie sich bitte, daß GOTO hier Branch heißt, wofür man aber nur B schreibt. Ganz entscheidend ist nun, daß (im Gegensatz zu den »8-Bitern«) es auch Branch-Befehle gibt, die mehrere Flags auf einmal berücksichtigen. Es gibt zusätzlich verschiedene Branch-Befehle für vorzeichenlose und vorzeichenbehaftete Zahlen (2er-Komplement). Natürlich gibt es auch Befehle, die nur auf ein Bit reagieren. Tabelle 3 bringt eine Übersicht.

Die in der Tabelle mit »*« gekennzeichneten Operatoren gelten nur für Zahlen im sogenannten 2er-Komplement-Format, das sind diese, bei denen das höchstwertige Bit als Vor-

zeichen dient. Die Befehle fangen immer mit B (wie Branch) an, gefolgt von zwei Buchstaben, die die Kurzform für die Bedingung sind. Wenn Sie nun beispielsweise BEQ (springe wenn gleich) schreiben, dann hängt es ausschließlich vom Z-Flag ab, ob der Befehl ausgeführt wird, oder nicht.

Das Z-Flag kann aber durch einen mehr oder weniger weit vor dem BEQ liegenden Befehl beeinflußt worden sein. Wenn Sie nun genau wissen, welcher Befehl das Z-Flag wie beeinflußt, dann könnten Sie das Risiko eingehen. Sicherer ist es auf jeden Fall, direkt vor dem BEQ einen Befehl zu schreiben, der das prüft. Wenn ich zum Beispiel springen will, wenn das Register D0=0 ist, dann schreibe ich:

```
CMP  #0,D0
BEQ  Marke
```

Der CMP-Befehl subtrahiert den Quelloperanden vom Zieloperanden, ändert je nach Ergebnis die Flags, schreibt aber nicht das Ergebnis ins Ziel. Der Vergleichsbefehl wirkt also auf die Flags wie eine Subtraktion. Das müssen sich die armen Kollegen mit den »8-Bitern« immer vor Augen führen, wenn Sie danach einzelne Flags (mit je einem Befehl) testen. Sie haben es besser. Sie dürfen zum Beispiel schreiben BGE (Springe, wenn größer oder gleich). Sie müssen sich nur dreierlei merken:

- diese Luxus-Befehle sind nur direkt nach einem CMP korrekt wirksam,
- der zweite Operand wird gegen den ersten verglichen. Wenn ich zum Beispiel springen will, wenn D0 größer als 9 ist (D0>9), schreibe ich:

```
CMP  #9,D0
BGT  Marke
```

- Man muß wissen, ob man die Operanden als vorzeichenbehaftete oder vorzeichenlose Zahlen vereinbart hat. Sinngemäß kann man die Kürzel auch im Zusammenhang mit DBcc anwenden, DBMI oder DBGT wären zwei Beispiele. BRA ist ein Sonderfall (springe immer); dem entspricht auch DBRA oder DBF.

Mit Listing 10 komme ich wieder zur Praxis. Die Funktion Read legt einen Text Zeichen für Zeichen in einem Puffer ab. Nun will ich wissen, welche Taste welchen Code erzeugt. Dazu muß ich den Inhalt des Puffers ausdrucken, und zwar in hex. Die Hauptaufgabe von Listing 10 ist somit die Ausgabe von Bytes in Form von je zwei Hex-Zeichen. Für ein A wäre zum Beispiel 41 zu drucken. Nun muß ich Ihnen allerdings gleich verraten, daß wir mit der normalen Console, mit der wir bisher gearbeitet haben (CON:) zwar die Standard-Tasten erfassen, nicht aber die Sondertasten, wozu die Cursor-Steuerung und die Funktionstasten zählen. Um also in einem Programm auf die Funktionstasten reagieren zu können, müssen wir etwas Besonderes tun. Zuerst dürfen wir nicht mehr die Standard-Handles Input und Output verwenden, sondern müssen selbst etwas für das I/O tun. Beim Amiga heißt das auch: Wenn wir nicht im CLI-Fenster arbeiten, müssen wir unser eigenes Fenster anlegen.

Dazu bietet uns der Amiga die Auswahl unter einigen Devices (Geräten) an. Dazu zählen unter anderem:

CON:	Console
RAW:	Console pur

Der Unterschied zwischen »CON:« und »RAW:« ist, daß nur letzteres alle Tasten (also auch die Sondertasten) behandelt. Der Nachteil von RAW: ist, daß alle Editierfunktionen ausge-

Kürzel der »Condition Codes«

Kürzel	Bedeutung	Deutsch
CC	Carry Clear	Carry = 0
CS	Carry Set	Carry = 1
EQ	Equal	Z = 1
GE	Greater or Equal	>=
*** GT	Greater Than	>
HI	Higher	>
***LE	Less or Equal	<=
LS	Less or Same	<=
***LT	Less Than	<
MI	Minus	—
NE	Not Equal	<>
PL	Plus	+
***VC	oVerflow Clear	V = 0
***VS	oVerflow Set	V = 1

Tabelle 3. Die Kürzel der Condition-Codes

```
        opt     1-                      ;nicht linken!

* F1 Funktionstasten lesen

        include OpenDos.i

_LVOOpen        equ     -30
_LVOClose       equ     -36

                move.l  #name,d1        ;Name von RAW:
                move.l  #1005,d2        ;Status = gibt es
                jsr     _LVOOpen(a6)    ;nun oeffnen
                move.l  d0,d5           ;Handle merken
                tst.l   d0              ;Fehler?
                beq     fini            ;wenn ja, abbrechen

loop            move.l  d5,d1           ;von RAW lesen
                move.l  #buffer,d2      ;in diesen Puffer
                move.l  #1,d3           ;Länge, siehe Text
                jsr     _LVORead(a6)    ;Lesen aufrufen
                cmp.b   #'x',buffer     ;Zeichen = 'x' ?
                beq     fertig          ;wenn ja

                move.l  buffer,d2       ;Zeichen nach d2
                lea.l   hbuf,a0         ;Ziel fuer Wandlung
                move.b  #' ',2(a0)      ;Blank nach Hex
                bsr     hex             ;in Hex-Zahl wandeln

* nun Hex-Zahl ausgeben:

                move.l  d5,d1           ;auch Output im Window
                move.l  #hbuf,d2        ;Address Hex-String
                move.l  #3,d3           ;Laenge
                jsr     _LVOWrite(a6)   ;Funktion "Schreiben"

                bra     loop            ;auf ein Neues

fertig          move.l  d5,d1           ;RAW wieder schliessen
                jsr     _LVOClose(a6)

* Zum Schluss immer die Lib schliessen!

                move.l  a6,a1           ;DOS-Lib-Basis
                move.l  _SysBase,a6     ;Basis Exec
                jsr     _LVOCloseLibrary(a6) ;Funktion "Schliessen"

fini            rts                     ;Return zum CLI

* Konvertiere d2.l in ASCII-String ab (a0)

hex
                moveq   #2-1,d1         ;nur fuer 2 Nibble (von 8)
next            rol.l   #4,d2           ;hole 1 Nibble
                move.l  d2,d3           ;nach d3 retten
                and.b   #$0f,d3         ;maskiere es
                add.b   #48,d3          ;in ASCII wandeln
                cmp.b   #58,d3          ;ist es >9 ?
                bcs     out             ;wenn nicht
                addq.b  #7,d3           ;sonst muss es A-F sein
out             move.b  d3,(a0)+        ;1 Zeichen abspeichern
                dbra    d1,next         ;next nibble
                rts

* Datenbereich:

dosname         dc.b    'dos.library',0
                cnop    0,2
name            dc.b    'RAW:40/100/580/80/Stop mit x',0
                cnop    0,2

buffer          ds.b    8
                cnop    0,4

hbuf            ds.b    10
```

Listing 10. So erfassen Sie die Funktionstasten

schlossen, sprich vom Anwenderprogramm selbst zu stellen sind. RAW: ist das typische Device für Editor-Programme.

Nun zur Aufgabe: Über dem CLI-Window soll ein neues (unser) Window angelegt werden. In diesem Fenster erscheinen Hexzahlen immer, wenn wir eine Taste tippen. Die Zahlen beschreiben die Codes der Tasten. Tippen Sie also die Buchstaben von a bis w, werden die zugehörigen ASCII-Codes von $41 bis $77 ($ heißt hex) ausgegeben.

Tippen Sie hingegen die Funktionstasten, passiert schon mehr. Jede Taste erzeugt drei Zeichen, und zwar:

F1:	9B 30 7E
F2:	9B 31 7E
F3:	9B 32 7E
...	

Betätigen Sie die Cursortasten, generiert das immer zwei Zeichen:

Auf:	9B 41
Ab:	9B 42
Links:	9B 43
Rechts:	9B 44

Allen Sondertasten ist also gemeinsam, daß sie eine Sequenz erzeugen, die mit $9B startet. Damit kann man recht einfach diese Tasten von allen anderen unterscheiden. Gleichzeitig ist das allerdings auch die Aufforderung, ein weiteres Zeichen auszuwerten, denn erst da beginnen die Unterschiede. Auf diese Feinheit verzichten wir vorerst und sagen einfach: Alles anzeigen, was kommt. Nun aber zum Listing 10.

Auch auf ein Gerät greifen wir wie auf einen File zu, müssen das Device also mit Open öffnen und später auch wieder schließen. Die zugehörigen DOS-LVO's finden Sie am Anfang des Listings. Zum Öffnen müssen wir den Namen des Gerätes (des Files) übergeben und den Access Mode. In den Include-Files findet man die Equates

```
MODE_OLDFILE    EQU  1005
MODE_NEWFILE    EQU  1006
```

Gemeint ist damit, ob man auf eine schon existierende Datei zugreifen oder eine neue anlegen will. Da wir wissen (hoffen), daß RAW: existiert, wählen wir 1005. Unter »name« kommt nun die große Überraschung:

```
name      dc.b    'RAW:40/
100/580/80/Stop mit x',0
```

Das reicht, um das Gerät RAW: zu öffnen. Gleichzeitig schaffen wir damit ein Window mit diesen Eigenschaften:

40/100:	linke, obere Ecke (x,y in Bildschirmpunkten)
580:	Breite des Windows
80:	Höhe des Windows
Stop mit x:	Window-Titel

Die Funktion gibt wie üblich eine Handle zurück, die wir sowohl für die Eingabe als auch für die Ausgabe benutzen können. Das läuft nun wieder über die schon bekannten Funktionen Read und Write. Bei Read müssen Sie aufpassen. Auch wenn Sie als Länge 3 angeben, werden für die Funktionstasten drei Reads erforderlich (die Länge 1 tut's auch).

Nach dem Lesen eines Zeichens folgt

```
cmp.b      #'x',buffer
;Zeichen = 'x' ?
beq         fertig
;wenn ja
```

und damit hätten wir unser erstes »IF THEN«. Im Klartext heißt das: Vergleiche (cmp = compare) die Konstante x mit dem ersten Byte im Puffer. Bei Übereinstimmung springe zur Label »fertig« (branch if equal = springe wenn gleich). Unser Programm läuft dadurch in einer Schleife, bis ein x eingegeben wird.

Die folgenden Zeilen bereiten die Hex-Wandlung vor. Das Unterprogramm »hex« erwartet ein Langwort in d2. Es legt das Ergebnis im Puffer hbuf ab.

Das Unterprogramm zeigt recht gut, daß man in Assembler oft auf Bit-Ebene arbeiten sollte. Damit läßt sich zum Beispiel die Hex-Konvertierung wesentlich einfacher lösen, als mit der klassischen Methode (fortlaufende Divison durch 16). Sie werden bei der Gelegenheit auch sehen, warum die Hex-Darstellung so vorteilhaft ist. Ein Beispiel: das Langwort besteht aus den 4 Byte des Inhalts AA, BB, CC, DD. Ich sehe sofort, daß im höherwertigen Wort AABB und im niederwertigen CCDD steht. In dezimal (2864434397) dürfte das schwerfallen. Das Problem der Routine »hex« ist nun, daß Sie tatsächlich ASCII-Zeichen ausgeben muß. Hat dort ein Digit den Wert Null, muß ich den ASCII-Code 48 (dezimal) ausgeben, um die »0« auf dem Schirm zu sehen. Das klappt ganz gut bis zur Neun (57), doch für 10 muß ich hex »A« drucken, und das hat den ASCII-Code 65, B hat 66 etc. Diese Lücke zwischen »9« und »A« müssen wir also füllen.

Zweites Problem: die Hexzahl sei $12345678 ($ heißt hex). $1 ist ein Nibble (Halbbyte), das im Register 4 Bit belegt (0001 binär). Natürlich muß ich $1 zuerst ausgeben, doch dafür muß ich $1 auf den Platz von $8 bringen, weil im Puffer das Zeichen (nach der Umwandlung in die ASCII-1) vorn stehen muß. Das Byte hat aber 8 Bit, 4 übertrage ich, die übrigen 4 Bit haben Werte, die nur stören, folglich muß ich sie ausblenden. Damit ergibt sich folgender Ablauf:

1. Das Nibble $1 auf den Platz von $8 bringen.
2. Dort die übrigen 4 Bit des Bytes auf Null setzen.
3. Das Nibble in ASCII wandeln.
4. Das Zeichen im Puffer ablegen.
5. Wiederhole 1.) bis 4.) mit den Nibbles $2, $3...$8.

Dazu eine Anmerkung: das Unterprogramm ist universell und kann auch Langworte konvertieren. Weil ich hier aber nur ein Byte (das höchstwertigste Byte!) (zwei Nibbles) ausgeben will, lasse ich die Schleife nur bis 2-1 laufen. Schritt 1 wird mit dem ROL-Befehl erledigt (Rotate Left). Wir benutzen von ROL die Syntax

```
ROL   #4,d2
```

Das heißt rotiere den Inhalt von d2 um 4 Bit nach links. Und was heißt nun rotieren? Nehmen wir an, in den d2 stehen diese 32 Bit:

vorher:	1111	0000	0000	0000	0000	0000	0000	0000
nach ROL #4	0000	0000	0000	0000	0000	0000	0000	1111

Die Bits werden nach links geschoben, und die Bits, die dann ganz links »herausfallen«, werden rechts wieder eingespeist.

Anders sähe es beim ASL-Befehl (Shift left, schiebe nach links) aus. Da wird auch nach links geschoben. Rechts werden Nullen eingespeist und links fallen die Bits heraus. Nun, wir haben ROL gewählt, und damit unser Ziel erreicht. Das Nibble (so nennt man ein Halb-Byte), das vorher ganz links stand, steht nun ganz rechts.

Unsere Schleife soll zweimal durchlaufen werden, folglich muß ich vorab (wegen des -1) den Schleifenzähler D1 mit 1 (2-1) laden. Nun erfolgt das berühmte ROL. Danach steht das Nibble am richtigen Platz, aber ausgeben kann ich leider nicht diese 4 Bit alleine, ich brauche ein Byte für ein ASCII-Zeichen, also 8 Bit.

Die 4 Bit links von meinem Nibble haben aber einen Wert, und der muß weg, genau: die 4 höherwertigen Bit des Bytes müssen 0000 werden. Das geschieht über die Maske $F mit »andi.b #$0F,d3«.

Beispiel:

in d3 steht	1011	1010
AND Maske	0000	1111
ergibt	0000	1010

Das Verfahren lebt davon, daß logisch UND nur dann wahr (1) ergibt, wenn alle Eingangsgrößen wahr sind. In Assembler wirkt UND (and) Bit für Bit. Nachdem wir so den reinen Zahlenwert (0..15 dezimal) isoliert haben, beginnt die Konvertierung in ASCII. 0 bis 9 (als Zahl) kann direkt in 0 bis 9 (als ASCII-Zeichen) umgesetzt werden, das sind die ASCII-Codes 48 bis 57. Das bedeutet auch, 48 müssen wir mindestens addieren. Nun vergleichen wir D3 mit 58 (eine hex 10, die man als A ausgeben muß). Ist die Zahl < 10 (also 0 bis 9), kann alles so bleiben, es geht zur Ausgabe. Andernfalls müssen wir die Lücke in der ASCII-Tabelle (65 für A-58=7) addieren.

An dieser Stelle ein Hinweis: Man sieht manchmal, daß erst geprüft wird, ob der Wert zwischen 0 und 9 liegt und dann im zweiten Test, ob es ein Wert zwischen 10 und 15 ist. Diese Methode ist in Ordnung, wenn die Zeichen aus einer Eingabe stammen, wo ja der Anwender auch ungültige Zeichen (nicht 0...9, A...F) eingeben kann. Wir holen hier aber Zahlen aus einem Register, da kann so etwas nicht drin stehen!

Die Befehle CMP und BCS kennen wir schon. Warum aber BCS? Ja eigentlich nur, um Ihnen zu zeigen, was andere Leute so schreiben (und denken müssen), die von CPUs kommen, die nicht so komfortabel wie der 68000 sind. Ein Vergleich wirkt auf die Flags wie eine Subtraktion. Bei einem »Borgen« wird auch das Carry-Flag gesetzt. Das ist aber der Fall, wenn D3 > 58 ist.

Nun ist es ja üblich, daß auf den Druck einer Funktions-Taste hin, ein Programm etwas mehr tut, sprich eine Routine aufruft.

Das können sehr komplizierte Programmteile sein, wir wollen aber das Prinzip üben in der Art von

```
IF F1 THEN GOTO...
IF F2 THEN GOTO...
...
```

Die aufgerufene Routine soll deshalb auch nur »Hier ist F1, F2 usw.« ausgeben, und dann auch schon bei F4 enden. Listing 11 zeigt die Lösung.

Ich glaube, das Programm muß ich nicht mehr groß erklären; aber jedes Programm hat einen Sinn. Notfalls kann es nämlich als abschreckendes Beispiel dienen, und das soll es auch. Stellen Sie sich vor, ein Programm mit 100 Kommandos oder mehr würde so arbeiten. Das wäre ein Umstand und ein Spaghetti-Code!

Neu ist nur, daß ich jetzt auf $9B prüfe und im Falle »keine Sondertaste« gleich auf das nächste Zeichen warte. Weil ich aber im Falle $9B noch eine Taste lesen muß, habe ich das Lesen in das Unterprogramm »GetKey« verlagert.

Viel eleganter löst man so etwas mit einem »ON X GOSUB«. Dazu müssen wir zwar wieder einmal den Schwierigkeitsgrad etwas steigern, aber wenn Sie das gelernt haben, ist eigentlich schon fast alles gelaufen. Jedes Programm besteht nämlich aus einer Hauptschleife, in der es auf Kommandos wartet. Die Kommandos werden interpretiert und die passenden Unterprogramme aufgerufen. Danach geht es weiter in der Hauptschleife. In Basic sähe das so aus:

```
10      INPUT  KOMMANDO
20      ON KOMMANDO GOSUB
        100,200,300, ...
30      GOTO 10
```

Wie das in Assembler aussieht, zeigt Listing 12.

Den Anfang kennen Sie schon fast. Nur wird hier der Code der Funktionstasten (das Byte nach $9B) durch Subtraktion von $30 in die Zahlen 0 bis 9 gewandelt, so steht er dann im Register D0. Der Einfachheit halber bearbeiten wir auch hier nur die Funktionstasten F1, F2, F3 und F4, also 0, 1,2 und 3 in D0.

Bitte beachten Sie, daß alle Tests fehlen (es geht um das Prinzip!), bei Betätigung aller anderen Tasten also das Programm abstürzt. Wir wollen ein »ON D0 GOSUB« realisieren und brauchen dazu natürlich zuerst vier Unterprogramme, die hier wieder F1, F2, F3 und F4 heißen. Diese Unterprogramme sind trivial. Sie geben nur die Meldungen »hier ist ..« aus. Nun betrachten wir das Listing wieder von unten. Da steht die Marke »table« und das, genau das, ist das Geheimnis unseres »ON X GOSUB«. Diese Tabelle ist eine Liste mit den Adressen der Unterprogramme.

Das Erstellen der Tabelle ist recht einfach. Schreiben Sie für jede Adresse

```
dc.l            Label
```

wobei für Label die Marke (die symbolische Adresse) des jeweiligen Unterprogramms einzutragen ist. Wichtig ist die Reihenfolge. Die Folge der Tasten muß sich in der Reihenfolge der Einträge der Tabelle wiederholen. Die Unterprogramme selbst können in beliebiger Reihenfolge im Listing stehen. Es besteht also eine enge Zuordnung zwischen den Kommandos (hier den Funktionstasten) und der Sprungtabelle. Deshalb kann man aus den Kommandos die zugehörige Adresse berechnen. Eine Adresse belegt immer 4 Byte.

Folglich gilt:

```
Platz = Beginn + 4 x
Tastennummer.
```

So einfach ist das also. Multiplizieren wir nun D0 mit 4. Dafür hat der 68000 natürlich auch einen speziellen Befehl (MULU), aber genau den nehmen wir hier nicht. Ein anständiger Assembler-Programmierer wird nämlich bei einer Multiplikation mit 2 oder 4 oder 8 oder 16 (Sie merken es: bei jeder 2er-Potenz) sofort hellhörig und greift auf einen Befehl zu, der das viel schneller erledigt. Hier heißt dieser Befehl ASL (Arithmetic Shift Left).

»ASL #1,d0« zum Beispiel schiebt alle Bits in D0 um eine Stelle nach links. Die Wirkung ist die gleiche, wie die im Dezimalsystem, wo Sie durch Linksschieben der Zahlen (und Festhalten des Kommas) mit 10 multiplizieren. Hier sind wir aber im dualen Zahlensystem, womit sich nur eine Multiplikation mit 2 ergibt. Schieben wir aber um zwei Stellen, so hätten wir schon unser »mal 4«. Das Ergebnis müssen wir auf den Beginn der Tabelle addieren. De-

```
        opt     1-              ;nicht linken!

* F2 Funktionstasten lesen und agieren

        include OpenDos.i

_LVOOpen        equ     -30
_LVOClose       equ     -36
        move.l  #name,d1        ;Name von RAW:
        move.l  #1005,d2        ;Status = gibt es
        jsr     _LVOOpen(a6)    ;nun oeffnen
        move.l  d0,d5           ;Handle merken
        tst.l   d0              ;Fehler?
        beq     fini            ;wenn ja, abbrechen

        lea.l   buffer,a3       ;Adresse des Puffers

loop    jsr     GetKey          ;Lese Taste
        cmp.b   #$9B,(a3)       ;Funktionstaste?
        bne     loop            ;wenn nicht
        jsr     GetKey          ;sonst lese Code

        cmp.b   #$30,(a3)       ;Taste F1 ?
        beq     F1              ;wenn ja
        cmp.b   #$31,(a3)       ;F2 ?
        beq     F2              ;wenn ja
        cmp.b   #$32,(a3)       ;F3 ?
        beq     F3              ;wenn ja
        cmp.b   #$33,(a3)       ;Taste F4
        beq     F4              ;wenn ja, fertig
        bra     loop

F1      lea.l   f1_text,a0      ;Adresse Text
        bsr     print           ;drucken
        bra     loop            ;auf ein Neues

F2      lea.l   f2_text,a0      ;Adresse Text
        bsr     print           ;drucken
        bra     loop            ;auf ein Neues

F3      lea.l   f3_text,a0      ;Adresse Text
        bsr     print           ;drucken
        bra     loop            ;auf ein Neues

F4      move.l  d5,d1           ;RAW wieder schliessen
        jsr     _LVOClose(a6)

* Zum Schluss immer die Lib schliessen!

        move.l  a6,a1                   ;DOS-Lib-Basis
        move.l  _SysBase,a6             ;Basis Exec
        jsr     _LVOCloseLibrary(a6)    ;Funktion "Schliessen"

fini    rts                     ;Return zum CLI

GetKey  move.l  d5,d1           ;von RAW lesen
        move.l  a3,d2           ;in diesen Puffer
        move.l  #1,d3           ;1 Zeichen
        jsr     _LVORead(a6)    ;Lesen aufrufen
        rts

print   clr.l   d3
        move.b  (a0)+,d3        ;Laenge
        move.l  d5,d1
        move.l  a0,d2
        jsr     _LVOWrite(a6)   ;Funktion "Schreiben"
        rts

* Datenbereich:

dosname dc.b    'dos.library',0
        cnop    0,2
name    dc.b    'RAW:40/100/580/80/Stop mit F4',0
        cnop    0,2

f1_text dc.b    8,'Hier F1',10
        cnop    0,2
f2_text dc.b    8,'Hier F2',10
        cnop    0,2
f3_text dc.b    8,'Hier F3',10
        cnop    0,2

buffer  ds.b    8
        cnop    0,4
hbuf    ds.b    10
```

Listing 11. Verzweigung mit vielen »IF...THEN«-Strukturen

```
          opt      1-                     ;nicht linken!
* F3      ON Funktionstasten GOSUB

          include OpenDos.i

_LVOOpen  equ  -30
_LVOClose equ  -36

          move.l   #name,d1               ;Name von RAW:
          move.l   #1005,d2               ;Status = gibt es
          jsr      _LVOOpen(a6)           ;nun oeffnen
          move.l   d0,d5                  ;Handle merken
          tst.l    d0                     ;Fehler?
          beq      fini                   ;wenn ja, abbrechen

          lea.l    buffer,a3              ;Adresse des Puffers

loop      jsr      GetKey                 ;Lese Taste
          cmp.b    #$9B,(a3)              ;Funktionstaste?
          bne      loop                   ;wenn nicht
          jsr      GetKey                 ;sonst lese Code

          move.b   (a3),d0                ;Code -> d0
          ext.w    d0                     ;auf Wort erweitern
          sub.w    #$30,d0                ;Code in 0..3
          asl.w    #2,d0                  ;mal 4
          lea.l    table,a0               ;Zeiger auf Tabelle
          move.l   0(a0,d0.w),a0          ;Adresse -> a0
          jsr      (a0)                   ;Routine aufrufen

          bra      loop                   ;bis F4 kommt

F1        lea.l    f1_text,a0             ;Adresse Text
          bsr      print                  ;drucken
          rts

F2        lea.l    f2_text,a0             ;Adresse Text
          bsr      print                  ;drucken
          rts

F3        lea.l    f3_text,a0             ;Adresse Text
          bsr      print                  ;drucken
          rts

F4        move.l   (sp)+,d0               ;Kill Return Adress
          move.l   d5,d1                  ;RAW wieder schliessen
          jsr      _LVOClose(a6)

* Zum Schluss immer die Lib schliessen!

          move.l   a6,a1                  ;DOS-Lib-Basis
          move.l   _SysBase,a6            ;Basis Exec
          jsr      _LVOCloseLibrary(a6)   ;Funktion "Schliessen"

fini      rts                             ;Return zum CLI

GetKey    move.l   d5,d1                  ;von RAW lesen
          move.l   a3,d2                  ;in diesen Puffer
          move.l   #1,d3                  ;1 Zeichen
          jsr      _LVORead(a6)           ;Lesen aufrufen
          rts

print     clr.l    d3
          move.b   (a0)+,d3               ;Laenge
          move.l   d5,d1
          move.l   a0,d2
          jsr      _LVOWrite(a6)          ;Funktion "Schreiben"
          rts

* Datenbereich:

table     dc.l     F1
          dc.l     F2
          dc.l     F3
          dc.l     F4

dosname   dc.b     'dos.library',0
          cnop     0,2
name      dc.b     'RAW:40/100/580/80/Stop mit F4',0
          cnop     0,2

f1_text   dc.b     8,'Hier F1',10
          cnop     0,2
f2_text   dc.b     8,'Hier F2',10
          cnop     0,2
f3_text   dc.b     8,'Hier F3',10
          cnop     0,2

buffer    ds.b     8
          cnop     0,4

hbuf      ds.b     10
```

Listing 12. »ON X GOSUB Y« – in Assembler kein Problem

ren Startadresse beschaffen wir uns mit »lea table,a0«. Nun kommt ein ganz wilder Befehl, nämlich

```
move.l    0(a0,d0.w),a0
```

Wir benutzen die Adressierungsart »ARI mit Index und Offset«. Nur »Index« gibt es leider nicht, also setzen wir das Offset zu null. Demnach errechnet sich die effektive Adresse als die Summe von a0 und d0. Die müssen wir nun in das Zielregister »moven«, und da nehmen wir gleich wieder a0. So etwas ist beim 68000 erlaubt, und weil es schön »tricky« aussieht, schreibt es auch jeder so. A0 zeigt nun also auf die Adresse des zugehörigen Unterprogramms, und das können wir nun schlicht mit »jsr (a0)« aufrufen.

Nach dem »jsr« kehrt das Programm sofort zu dem »jsr« folgenden Befehl zurück. Der heißt »bra loop«, also beginnt es wieder von vorn mit dem nächsten Kommando.

Die Ausnahme von dieser Regel finden Sie im Unterprogramm »F4«.

Dieses Unterprogramm ist gar keines, es wird zwar mit JSR aufgerufen, es endet aber nicht mit RTS. Folglich müssen wir die noch auf dem Stack befindliche Return-Adresse entfernen.

Wir erledigen das an dieser Stelle mit dem Befehl »move.l (sp)+,d0«. Das ist hier zulässig, weil in diesem Fall D0 nicht mehr benötigt wird.

Ist doch ganz einfach, oder? Es ist Ihnen zu einfach? Nun gut, machen wir die Sache etwas komplizierter. Unser schöner Kommando-Interpreter hat einen Nachteil. Die Kommandos müssen in der Reihenfolge von F-Tasten-Codes auftreten. Auch andere Folgen, wie 1 bis 9 oder A bis M sind denkbar, es muß aber immer eine Folge sein. Sie wissen jetzt, warum manche Leute ein Menü anbieten der Art:

```
1 = Eingabe
2 = Rechnen
3 = Stoppen
```

Das merkt sich schlecht, besser wäre doch

```
E = Eingabe
R = Rechnen
S = Stoppen
```

Das Prinzip ist natürlich wieder ganz einfach. Es gibt zwei Tabellen. In der ersten Tabelle stehen die »Keys« (die erlaubten Tasten oder Kommandos), in der zweiten die Adressen der zugehörigen Routinen. So muß man doch nur den Key in der ersten Tabelle suchen und aus seiner Platznummer in dieser Tabelle einen Zeiger auf den richtigen Platz in der Adreßtabelle errechnen. Da kann man dann die Adresse herausholen und ab geht's.

Diesmal soll unser Programm aber wasserdicht sein. Folglich muß der Fall »nicht gefunden« abgefangen werden. Wir wollen auch den User nicht zwingen, immer die Shift-Taste zu betätigen. Deshalb sollen Groß- und Kleinbuchstaben gleich behandelt werden. Schließlich soll das Programm universell sein, sprich: es muß mit minimalem Aufwand möglich sein, Funktionen hinzuzunehmen oder zu ändern.

Nun denn, hier ist Listing 13. Wir lesen wieder eine Taste, jetzt aber nur eine, weil wir keine Funktionstasten erwarten, beziehungsweise diese ignorieren werden.

Nachdem das Zeichen in das Register d0 geladen wurde, soll es in einen Großbuchstaben gewandelt werden, falls es nicht schon ein Großbuchstabe ist. Force Uppercase nennen das die Fachleute in Neudeutsch.

Wenn Sie einmal auf eine ASCII-Tabelle schauen, wird Ihnen sicherlich auffallen, daß sich die Klein- und die Großbuchstaben immer um 32 unterscheiden. 2 hoch 5 ist aber auch 32, sprich, im Falle von Kleinbuchstaben ist Bit 5 gesetzt. Folglich wird ein guter (sind Sie doch) Assembler-Programmierer nicht sagen »wenn der Code > Z ist, dann subtrahiere 32«, sondern er sagt ein-

fach »lösche Bit 5«. Auch dafür kennt der 68000 einen Befehl, nämlich »BCLR« (Bit Clear).

```
bclr #5,d0
```

löscht Bit 5 (setzt es auf 0) im Operanden (hier d0).

An dieser Stelle sind wir also soweit, daß wir prüfen können, ob der User E, R oder S eingegeben hat. Wenn Sie nun bitte einmal auf das Ende des Listings schauen: Da gibt es eine kleine Tabelle für diese drei Zeichen. Darunter aber steht:

```
count equ *-keys
```

Wow, schon wieder so ein Ding! Fangen wir mit »equ« an. »equ« ist ein sogenanntes Equate (englisch), Sie lesen es aber am besten als »equal = gleich). Die Assembler-Direktive von zum Beispiel

```
Anton equ 4711
```

bedeutet, ab jetzt kann man anstatt 1 auch Anton sagen. »JSR Anton« wäre dann ein gültiger Befehl. Prinzipiell ist das nichts weiter als reine Textverarbeitung. Der Assembler setzt einfach nachher für Anton immer eine 4711 ein. Wir müssen Anton aber jetzt auch als Konstante ansehen. Falls Sie Pascal können, betrachten Sie das »equ« wie »const«, als C-Programmierer wie ein »#define«. Daher dürfen wir in diesem Programm auch beim Bezug auf »count« nicht »count« schreiben, »#count« ist Pflicht!

Das nächste »Wow« ist der »*«. Als erstes Zeichen in einer Programmzeile ist er ganz harmlos und bedeutet nur »Kommentar folgt«. Als Operand heißt er Location Counter (LC). Sie wissen, daß Befehle, je nach Anzahl und Art der Operanden verschieden viel Speicher belegen. Deshalb führt der Assembler einen Zähler, der sozusagen die bis zu jedem Befehl verbrauchten Byte mitzählt. In diesem Sinne entspricht der LC dem PC (Program Counter), mit dem nachher die CPU ein Programm verfolgt. Der Unterschied: Der LC wird auch durch Assembler-Direktiven inkrementiert, wie zum Beispiel »dc.b«, das ja auch Bytes (mit Daten) belegt.

In der Assembler-Syntax heißt nun aber der LC nicht LC sondern »*«. Betrachten wir ein Beispiel. Im Listing hätte der LC zu Beginn der Zeile mit der Marke »keys« den Wert 100. Die Anweisung »dc.b E, R, S » läßt ihn auf 100 (für E), stellt ihn dann auf 101 (für R) und schließlich auf 102 (für S).

Die nächste Zeile mit der Marke »count« sieht den LC als 103. An dieser Stelle erfolgt aber ein Equate. Da in Equates auch Ausdrücke erlaubt sind, hat

```
count    equ    *-keys
```

die Wirkung von

```
count = LC - keys
```

Das in Zahlen ergibt

```
count = 103-100
```

Damit hätten wir unsere symbolische Konstante 3. Warum habe ich da nicht einfach »count equ 3« geschrieben? Antwort: Das machen nur Anfänger! Wenn wir nämlich später die Tabelle erweitern, können wir das tun, ohne die Zeile mit dem »equ« anfassen zu müssen. In jedem Assemblerlauf (der muß dann eh sein) wird automatisch die richtige Anzahl eingesetzt. Außerdem: Tabellen können ganz schön lang sein, da kann man sich leicht verzählen. Nun müssen wir den Key (steht immer noch in D0) in der

```
           opt      l-                 ;nicht linken!
* CASE X OF

           include OpenDos.i

_LVOOpen   equ      -30
_LVOClose  equ      -36

           move.l   #name,d1           ;Name von RAW:
           move.l   #1005,d2           ;Status = gibt es
           jsr      _LVOOpen(a6)       ;nun oeffnen
           move.l   d0,d5              ;Handle merken
           tst.l    d0                 ;Fehler?
           beq      fini               ;wenn ja, abbrechen

           lea      buffer,a3          ;Adresse des Puffers

loop       jsr      GetKey             ;Lese Taste

* Key in Tabelle gueltiger Keys suchen
* -------------------------------------
           move.b   (a3),d0            ;Code -> d0
           bclr     #5,d0              ;force uppercase
           lea      keys,a0            ;Tab. gueltige Keys
           move     #count,d1          ;deren Anzahl
search     cmp.b    (a0)+,d0           ;Key hier?
           dbeq     d1,search          ;wenn nicht
           tst      d1                 ;Key gefunden?
           bmi      loop               ;wenn nicht

* Rechne Adresse zu Key
* --------------------
           neg      d1                 ;sub d1,#count
           add      #count,d1          ;ergibt Platz-Nr.
                                       von Key
           lsl      #2,d1              ;die mal 4
           lea      table,a0           ;Zeiger auf Tabelle
           move.l   0(a0,d1),a0        ;Adresse -> a0

           jsr      (a0)               ;Routine aufrufen

           bra      loop               ;bis F4 kommt
Eingabe    lea      E_text,a0          ;Adresse Text
           bsr      print              ;drucken
           rts

Rechnen    lea      R_text,a0          ;Adresse Text
           bsr      print              ;drucken
           rts

Stoppen    move.l   (sp)+,d0           ;Kill Return Adress
           move.l   d5,d1              ;RAW wieder schliessen
           jsr      _LVOClose(a6)

* Zum Schluss immer die Lib schliessen!
           move.l   a6,a1                 ;DOS-Lib-Basis
           move.l   _SysBase,a6           ;Basis Exec
           jsr      _LVOCloseLibrary(a6)  ;Funktion "Schliessen"

fini       rts                            ;Return zum CLI

GetKey     move.l   d5,d1                 ;von RAW lesen
           move.l   a3,d2                 ;in diesen Puffer
           move.l   #1,d3                 ;1 Zeichen
           jsr      _LVORead(a6)          ;Lesen aufrufen
           rts

print      move.l   a0,a1                 ;kopiere Textzeiger
p1         addq.l   #1,a1                 ;+1
           cmp.b    #0,(a1)+              ;Null-Byte?
           bne      p1                    ;wenn nicht
           sub.l    a0,a1                 ;= Textlaenge

           move.l   a1,d3                 ;Laenge
           move.l   d5,d1                 ;Handle
           move.l   a0,d2                 ;Adresse Text
           jsr      _LVOWrite(a6)         ;Funktion "Schreiben"
           rts

* Datenbereich:

table      dc.l         Eingabe
           dc.l         Rechnen
           dc.l         Stoppen

keys       dc.b         'E','R','S'
count      equ          *-keys
           cnop         0,4

dosname    dc.b         'dos.library',0
           cnop         0,2
name       dc.b         'RAW:40/100/580/80/Stop mit S',0
           cnop         0,2

E_text     dc.b         'Hier Eingabe',10,0
           cnop         0,2
R_text     dc.b         'Hier Rechnen',10,0
           cnop         0,2

buffer     ds.b         8
           cnop         0,4

hbuf       ds.b         10
```

Listing 13. Universelle Menütechnik

Tabelle »keys« suchen. Diesen Teil finden Sie unter »Suche Tasten-Code in Tabelle«. Das Problem: auch »nicht gefunden« muß gemeldet und erkannt werden. Die Lösung ist eine DBcc-Schleife. Führen Sie sich bitte immer vor Augen

```
DBcc          dn,loop
```

heißt: verlasse die Schleife, wenn die Bedingung cc erfüllt ist, oder dn auf -1 gelaufen ist, sonst springe zu »loop«.

Vorab wird mit »lea keys,a0« ein Zeiger auf den Beginn der Tabelle gestellt. Der Trick: Der Schleifenzähler D1 wird mit Count geladen (hier 3), eine DBcc-Schleife läuft aber im Grenzfall bis -1, hier also über vier Schritte. Wenn die Schleife wegen des »eq« (»equal« bedeutet gefunden) im »dbeq« verlassen wird, kann D1 nicht negativ sein. Ist es negativ, kommt das »bmi start« zur Wirkung (springe, wenn negativ).

Unter »Suche Adresse zu Key« taucht nun das nächste Problem auf. In der DBcc-Schleife lief D1 ja rückwärts. Es hat also diese Werte im Falle von »gefunden von«:

Key	D1
1	3
2	2
3	1

Ich brauche aber die Folge 0, 1, 2, um wieder auf die Adreßtabelle wie im vorigen Beispiel zugreifen zu können. Dieses ergäbe sich ganz einfach, wenn ich von Count D1 subtrahieren würde. (3-3=0, 3-2=1, 3-1=2). Leider erlaubt der 68000 den Befehl »sub d1,#count« nicht; wie soll er auch von einer Konstanten etwas subtrahieren? Da hilft die Anweisung

```
neg d1
```

Ich negiere d1. War es 3, dann wird es -3. Darauf addiere ich Count, ergibt 0. Alte Regel: Wenn man nicht subtrahieren kann, muß man eben den negativen Wert addieren. Den Rest hatten wir schon. Als kleinen Unterschied habe ich hier nun anstatt »asl« »lsl« genommen (logical shift left«. Der Unterschied: ASL schiebt arithmetisch korrekt, würde also auch das Vorzeichen berücksichtigen, wenn wir eines hätten. Damit wären wir wieder an der Stelle angekommen, die das vorige Programm ausmachte. Mit »lea table,a0« zeigen wir auf die Adreßtabelle, und den Rest kennen Sie schon, fast...

Ein Blick auf die Texte zeigt Ihnen, daß ich die Längenbytes weggelassen habe. Statt dessen sind alle Texte mit einem Null-Byte abgeschlossen. Diese Technik ist sehr praktisch, weil ich mich jetzt um die Textlänge (das Abzählen der Zeichen) überhaupt nicht mehr kümmern muß. Nun muß allerdings das Unterprogramm etwas mehr tun, sprich in einer kleinen Schleife das Null-Byte suchen. Die Routine fängt nur einen Fall nicht ab, nämlich daß der String leer ist (nur aus einem Null-Byte besteht).

Makro ist die Kurzform von Makro-Befehl. Makro selbst heißt so viel wie »groß«. Prinzipiell ist ein Makro nur eine Zusammenfassung einer Gruppe von Einzelbefehlen, wie wir sie bisher kannten, unter einem neuen Namen. Leider hat jeder Assembler da seine eigene Syntax. Ich bringe deshalb alle folgenden Beispiele in der Makrosprache der HiSoft/Metacomco-Assembler (stimmen überein). Hier ein Beispiel:

```
CALLEXEC    macro
      move.l _SysBase,a6
      jsr    _LVO\1(a6)
             endm
```

Dieser Makro realisiert die Funktion CALLEXEC (Funktion der Exec-Library aufrufen), wie wir sie schon kennen. Jeder Makro hat einen Namen, der im Label-Feld stehen muß, gefolgt von dem Schlüsselwort »macro«. Ein Makro endet mit dem Schlüsselwort »endm«. Zwischen »macro« und »endm« kann eine beliebige Anzahl von Befehlen stehen. Ist der Makro einmal definiert, kann er beliebig oft mit seinem Namen aufgerufen werden. Innerhalb eines Makros dürfen auch die Namen anderer, dann schon vorher definierter Makros, stehen. Ob und wie tief Makros so geschachtelt werden dürfen, lesen Sie aber besser im Handbuch des verwendeten Assemblers nach.

Denken Sie jedoch immer an eines: Makros sind reine Textbausteine. Immer, wenn Sie den Makro aufrufen, wird lediglich dessen Text in den Programmtext eingefügt. Es handelt sich also nicht um Unterprogramme. Hier zwei Makros, die Sie zu Beginn eines Programms tippen können.

```
CALLLIB MACRO
        JSR     \1(A6)
        ENDM
```

```
                  opt 1-                   ;nicht linken!
_SysBase          equ 4                    ;Basis von Exec
_LVOOpenLibrary   equ -552                 ;Library oeffnen
_LVOCloseLibrary  equ -414                 ;Library schliessen
_LVOOutput        equ -60                  ;DOS: Output-Handle holen
_LVOWrite         equ -48                  ;Ausgabe
_main             move.l #dosname,a1       ;Name der DOS-Lib
                  moveq #0,d0              ;Version egal
   LINKLIB        OpenLibrary,_SysBase     ;DOS-Lib oeffnen
                  tst.l  d0                ;Fehler?
                  beq    fini              ;wenn Fehler, Ende
                  move.l  d0,_DOSBase      ;Zeiger merken
   LINKLIB        Output,_DOSBase          ;Hole Output-Handle
                  print  d0,#string,20     ;Text ausgeben
                  move.l  _DOSBase,a1      ;Basis der Lib
   LINKLIB        CloseLibrary,_SysBase    ;Funktion "Schliessen"
fini              rts                      ;Return zum CLI
_DOSBase          dc.l  0
dosname           dc.b  'dos.library',0
                  cnop  0,2
string            dc.b  'Hallo lieber Leser!',10
                  cnop  0,2
```

Listing 14. Der Einsatz von Makros bringt Arbeitserleichterung und reduziert den Schreibaufwand

```
LINKLIB MACRO
        MOVE.L  \2,A6
        CALLLIB \1
        ENDM
```

Der Makro »CALLLIB« wie »Call Library« beinhaltet das Ihnen schon bekannte »jsr offset(a6)«. Wichtig ist hier, daß wir dem Makro einen Parameter übergeben müssen, nämlich den einzusetzenden String. Für solche Parameter haben Makros Variable. Meistens üblich sind aber Ziffern mit einem Schrägstrich oder (bei SEKA) einem Fragezeichen davor. Bei Metacomco und DevPac 2.0 sind auch noch die Buchstaben A bis Z erlaubt. Beachten Sie bitte, daß der Schrägstrich bei Metacomco und HiSoft ein »Backslash« (nach links gekippter Strich) sein muß. Das Programm in Listing 14 soll lediglich das schon bekannte »Hallo« ausgeben.

Das sieht doch richtig gut aus. Was ist nun der Haken an der Sache? Es fehlen die Makros, die ich Ihnen mit Listing 15 vorstellen möchte.

Den Makro LINKLIB finden Sie in ähnlicher Form in den Include-Files von Metacomco und Hisoft.

Die Zeilen 2 bis 4 können Sie prinzipiell auch weglassen. Da Sie aber in den Include-Files häufig anzutreffen sind (schauen Sie mal rein, lohnt sich), sollte das aber doch erklärt werden. NARG ist eine Assembler-Variable und heißt »Number Arguments«. Diese Variable ist nur innerhalb eines Makros gültig (sonst Null) und hält die Anzahl der Parameter, mit denen der Makro aufgerufen wurde. Ob die Anzahl stimmt, ist eine andere Frage, aber das kann man prüfen. Dazu benutzt man eine zweite Eigenschaft guter Assembler, nämlich bedingtes Assemblieren. Hierfür gilt die generelle Form

```
IFcc
(tue das, wenn cc true)
ENDC
(hier weiter, wenn cc
false)
```

Die Bedingungen »cc« sind prinzipiell die gleichen, wie wir sie schon von den Branch-Befehlen her kennen. Die Einschränkung ist allerdings, daß immer nur ein Argument gegen Null verglichen werden kann. Will ich also prüfen, ob die Anzahl der Makro-Argumente (NARG) stimmt, und im Fehlerfall eine Warnung ausgeben, so muß ich sagen: Wenn NARG minus 2 nicht gleich Null oder

```
IFNE NARG-2
```

Sie brauchen diese Tests in aller Regel nicht durchzuführen, der Assembler meckert dann nur etwas später. Nun hätten wir noch ein kleines Problem. Nehmen wir an, Sie hätten einen Makro geschrieben der Art von

```
nonsens macro
loop    move     d1,d2
bra     loop
        endm
```

Schreiben Sie nun in Ihrem Programm

```
nonsens
nonsens
```

so wird der Makroprozessor daraus folgende Zeilen entwickeln:

```
loop          move     d1,d2
bra           loop
loop          move     d1,d2
bra           loop
```

Spätestens beim zweiten »bra loop« wird der Assembler ins Schleudern geraten. Zu welchem »loop« soll er denn nun

springen? Er wird mit einer Fehlermeldung aussteigen.

Um so etwas zu vermeiden, gibt es nun in guten Assemblern immer eine Lösung. Setzen Sie anstatt eines Namens für die Marke $n ein. Für n ist eine Zahl zwischen 1-999 einzusetzen. Diese Zahl wird bei jedem Aufruf des Makros um eins hochgezählt. Wenn Sie mehrere Makros mit internen Labels verwenden, müssen Sie allerdings darauf achten, daß Sie mit unterschiedlichen und weit auseinanderliegenden Zahlen starten.

Auch die Include-Anweisung kann man zuerst zusammen mit Makros sinnvoll einsetzen. Nehmen wir an, Sie haben die Makrodefinition von Listing 15 in einem Textfile mit dem Namen »Mac66« abgelegt. Dann

```
LINKLIB MACRO
                IFNE        NARG-2
                FAIL ---- Makro LINKLIB: Nicht 2 Argumente ----
                ENDC
                MOVE.L      A6,-(SP)
                MOVE.L      \2,A6
                JSR         _LVO\1(A6)
                MOVE.L      (SP)+,A6
                ENDM

print   MACRO
                IFNE        NARG-3
                FAIL ---- Makro print: Nicht 3 Argumente ----
                ENDC
                MOVE.L      \1,D1         ;Ausgabe-Handle
                MOVE.L      \2,D2         ;Address Text
                MOVEQ       #\3,D3        ;Laenge Text
                LINKLIB Write,_DOSBase ;Funktion "Schreiben"
                ENDM
```

Listing 15. Die Makros, die vom Listing 14 eingebunden werden

können Sie das Programm in Listing 14 jetzt so schreiben:

```
include "Mac66"

PRINT    #msg1,#len1
PRINT    #msg2,#len2

msg1       dc.b     'Hallo'
len1       equ      *-msg1
ds.w       0
msg2       dc.b     'Amiga'
len2       equ      *-msg2
end
```

Diese Methode ist sehr sinnvoll, denn Sie werden sicherlich in jedem Programm zahlreiche DOS-Funktionen einsetzen. Wenn Sie sich diese Funktionen einmal als Makros definiert und in Ihrem »Mac-File« abgelegt haben, ersparen Sie sich nicht nur eine Menge an Tipperei, sondern auch auch einiges an Zeit für die Fehlersuche wegen nicht gemachter Tippfehler. Es schadet durchaus nicht, wenn im jeweiligen Programm viele der Makros nicht genutzt werden. Sie erzeugen ja dann auch keinen Code. Wird die »Mac-Lib« (Kürzel für Library = Bibliothek) zu groß, kostet es natürlich Zeit, wenn sie der Assembler bei jedem Lauf einlesen muß. Auf einer RAM-Disk kann dies unter Umständen zu Platzproblemen führen. Deshalb sollten Sie Ihre »Lib« möglichst in mehrere kleine Files nach Sachgebieten aufteilen.

Daß Ihr Amiga ein Multitasking-Betriebssystem hat, wissen Sie natürlich, nur hat das auch in Assembler einige Konsequenzen. Prinzipiell kann Ihr Task (Programm) so tun, als sei er der einzige Task im System. Praktisch kommen Sie damit nicht weit, denn Sie müssen mit anderen Tasks kommunizieren, um zum Beispiel zu erfahren, ob die Maus bewegt wurde. Die Kommunikation erfolgt beim Amiga über Message-Ports. Gesteuert wird das Ganze vom Message-Dispatcher, den Sie sich wie das Mädchen in der Telefonzentrale vorstellen können, das die Verbindungen zwischen den einzelnen Teilnehmern herstellt. Nur haben wir eine Super-Telefonistin. Sind Sie nämlich gerade besetzt, schreibt das Mädchen alles auf und gibt Ihnen die Nachricht durch, sobald Sie wieder frei sind. Etwas fachlicher: Die Nachrichten werden in einem Message-Queue gepuffert. Sie müssen natürlich mindestens ein »Telefon« haben, sprich mindestens einen Message-Port für Ihren Task einrichten.

Typisch für einen Task ist, daß er auf eine Nachricht wartet, also darauf, daß der User endlich mal eine Taste drückt oder die Maus bewegt. Dahinter steckt: Alle Eingaben laufen über einen Task mit dem Namen »input.device«. Der Amiga sorgt dafür, daß Nachrichten zu Ihrem Task (Ihrem Fenster) gelangt, nur Sie müssen darauf warten.

Die unfeinste Methode ist nun das sogenannte Polling. Dazu geht der Task einfach in eine Schleife, in der er den Message-Port so lange abfragt, bis dort eine Nachricht anliegt. Damit verbraucht er nur Rechnerzeit, die anderen Tasks dann fehlt.

Besser ist es eine Funktion mit dem Namen Wait zu benutzen. In diesem Falle wird der Task aus der Liste der aktiven Tasks gestrichen und geht auf die Liste der wartenden Task, wo er den Betrieb nicht mehr aufhält. Man sagt auch, der Task schläft. Wach bleibt aber das Betriebssystem, das ständig prüft, ob eine Nachricht für den Task kommt. Ist das der Fall, wird der Task wieder geweckt (er setzt nach dem Wait-Aufruf fort) und kann nun seine Nachricht behandeln, zum Beispiel auf einen Tastendruck so reagieren, wie es das Programm vorsieht.

Screens, Windows und Gadgets

Die einfachste Form von Wait ist die Funktion WaitPort(). Damit wartet man auf einen Port. Nun kann der Task aber mehrere davon haben, nicht aber auf alle Ports warten wollen. Für diesen Zweck gibt es die Signal-Bits. Jeder Port wird einem von 32 Bits zugeordnet, jeder Task hat seine eigenen Signal-Bits. Ein Task kann allerdings nur maximal 16 Bit belegen, die anderen 16 braucht das System. Um nun auf eine beliebige Port-Kombination zu warten, muß man nur die entsprechenden Bits »verodern« (ihre Werte addieren) und damit Wait() aufrufen (im Gegensatz zu WaitPort). Ein Task sendet eine Message mit PutMsg(). Erreicht diese Nachricht einen schlafenden Task (einen der Wait() aufgerufen hatte), wird er damit geweckt. Der Empfänger-Task wird nun mit GetMsg() die Nachricht lesen. Er sollte dann mit ReplyMsg() den Empfang quittieren. Letzteres ist bei einigen Tasks, zum Beispiel Intuition, absolute Pflicht.

Das Thema Multitasking ist hiermit noch bei weitem nicht vollständig behandelt, doch wir müssen uns jetzt erst einmal um einige andere Amiga-Spezialitäten kümmern, um danach wieder automatisch beim Multitasking zu landen.

Neben den Libraries gibt es noch drei grundlegende Dinge, die man kennen sollte, nämlich Screens, Windows und Gadgets. Der Amiga erlaubt, mehrere virtuelle Bildschirme einzusetzen. Auf jedem Screen können sich wiederum mehrere Windows befinden. Der äußerliche Unterschied: Ein Screen kann immer nur vertikal verschoben werden, seine vorgewählte Größe kann nicht geändert werden. Einen Screen zu schaffen, ist recht einfach. Man definiert eine Struktur, in der die gewünschten Parameter eingetragen werden. Dann ruft man OpenScreen() auf. Die Funktion gibt einen Zeiger zurück oder NULL, wenn etwas schiefging. Dieser Zeiger muß dann als ein Parameter in die Window-Struktur eingetragen werden. Die wesentlichen Screen-Parameter sind die Auflösung (der Amiga kennt derer vier) und die Tiefe, womit die Anzahl der Bit-Planes gemeint ist. Mit einer Tiefe von zum Beispiel 3 sind $2^3 = 8$ Farben möglich.

Ein Window wird sinngemäß wie ein Screen geöffnet, nur daß jetzt hierfür eine Window-Struktur definiert werden muß. Generell ist das Window das Element, das am mächtigsten ist, und mit dem Sie wohl auch am meisten umgehen werden. Es ist zum Beispiel nicht unbedingt nötig, einen Screen zu öffnen. Trägt man nämlich in der Window-Struktur für den Parameter Screen NULL ein (und als Typ WBENCHSCREEN), wird automatisch der Workbench-Screen benutzt.

Nun aber gleich mit Listing 16 zur Praxis. Das Programm soll ein Window auf den Schirm bringen und in das Window einen Text schreiben (zeichnen). Das Fenster soll auf dem Bildschirm verschiebbar sein, seine Größe darf geändert werden. Das Programm soll enden, wenn die Close-Box des Fensters angeklickt wird.

Dazu öffne ich zwei Libraries, nämlich Intuition und Graphics. Generell braucht man immer beide, ihre Basis-Zeiger sollte man sofort sichern, die werden häufig benötigt. Die Exec-Library ist sowieso immer dabei. Das obligatorische

```
move.l  SysBase,a6
jsr     _LVOxxx(a6)
```

ist im Makro

```
CALLEXEC        xxx
```

versteckt.

Sinngemäß funktioniert der Makro CALLINT, der eine Funktion der Intuition-Library aufruft. Intuition ist für die komplexen

```
* window1

* In diesen Files stecken diverse Deklarationen und Makros.
* Schauen Sie mal rein!

        incdir  ":include/"

        include intuition/intuition.i
        include intuition/intuition_lib.i
        include exec/exec_lib.i
        include graphics/graphics_lib.i

* Intuition Library oeffnen:
* --------------------------
        lea             intname,a1
        moveq           #0,d0
        CALLEXEC        OpenLibrary
        tst.l           d0
        beq             abbruch
        move.l          d0,_IntuitionBase ;Basis-Zeiger sichern

* Ggraphics Library oeffnen
* -------------------------
        lea             grafname,a1
        moveq           #0,d0
        CALLEXEC        OpenLibrary
        tst.l           d0
        beq             closeint
        move.l          d0,_GfxBase     ;Basis-Zeiger sichern

* Window oeffnen
* --------------
        lea             windowdef,a0    ;zeige auf Window-Struktur
        CALLINT         OpenWindow      ;oeffne Window
        tst.l           d0              ;ging was schief?
        beq             closegraf       ;wenn ja
        move.l          d0,windowptr    ;Window-Zeiger sichern

* Text im Fenster zeichnen
* ------------------------
        moveq           #100,d0         ;X-Position
        moveq           #50,d1          ;Y
        move.l          windowptr,a1    ;Via Window-Zeiger
        move.l          wd_RPort(a1),a1 ;Rast-Port-Adresse holen
        CALLGRAF        Move            ;Funktion Move to X,Y

        move.l          windowptr,a1    ;brauche wieder Rastport
        move.l          wd_RPort(a1),a1
        lea             msg,a0          ;Adresse Text
        moveq           #msglen,d0      ;seine Laenge
        CALLGRAF        Text            ;und ausgeben

* Auf Event warten (kann hier nur WINDOWCLOSE sein)
* -------------------------------------------------
        move.l          windowptr,a0    ;zeige auf Window-Struktur
        move.l          wd_UserPort(a0),a0 ;nun auf Message-Port
        move.b          MP_SIGBIT(a0),d1 ;Anzahl Signal Bits -> d1
        moveq.l         #1,d0           ;in Maske
        lsl.l           d1,d0           ;konvertieren
        CALLEXEC        Wait            ;Schlaf gut!


* Fenster schliessen
* ------------------
        move.l          windowptr,a0    ;wir sind wieder wach
        CALLINT         CloseWindow     ;Fenster zu


* Libraries schliessen
* --------------------
closegraf
        move.l          _GfxBase,a1
        CALLEXEC        CloseLibrary

closeint
        move.l          _IntuitionBase,a1
        CALLEXEC        CloseLibrary

abbruch
        move.l          #0,d0           ;oder normales Ende
        rts

W_Gadgets       equ     WINDOWSIZING!WINDOWDRAG!WINDOWDEPTH!WINDOWCLOSE
W_Extras        equ     SMART_REFRESH!ACTIVATE

W_Title         dc.b    'Fenster-Titel',0

windowdef
                dc.w    200,50          ;links, oben
                dc.w    300,100         ;Breite, Hoehe
                dc.b    -1,-1           ;Pens des Screen
                dc.l    CLOSEWINDOW     ;einziges IDCMP Flag
                dc.l    W_Gadgets!W_Extras ;Window Flags
                dc.l    0               ;keine User-Gadgets
        dc.l            0               ;keine User-Checkmark
        dc.l            W_Title         ;Titel des Window
        dc.l            0               ;kein eigener Screen
        dc.l            0               ;keine Super Bitmap
        dc.w            100,20          ;Min. Groesse
        dc.w            640,200         ;Max.
        dc.w            WBENCHSCREEN    ;Use Workbench Screen

intname                 INTNAME         ;Name Intuition Lib
                                        (via Makro)
grafname                GRAFNAME        ;Name Graphics Lib

msg                     dc.b            'Hello, World! '
msglen                  equ             *-msg

_IntuitionBase          ds.l    1       ;Speicher fuer Zeiger
_GfxBase                ds.l    1
windowptr               ds.l    1
```

Listing 16. Ein erstes Intuition-Window

Dinge, wie Fenster zuständig, Graphics hingegen für die Grundroutinen (Zeichnen von Linien, Flächen oder Texten).

Um ein Window zu öffnen, reicht der einfache Aufruf, wie im Listing gezeigt. Alles Weitere steht in der Struktur ab Label »windowdef«. Als einziges IDCMP-Flag habe ich CLOSEWINDOW angegeben. Das heißt, nur wenn der User das Fenster schließt, meldet mir das Intuition. Alle anderen Events (Ereignisse) behandelt Intuition selbst. Natürlich kann man noch weitere Events zulassen, zum Beispiel MOUSEBUTTONS oder MOUSEMOVE, um nur einige zu nennen.

In der Folgezeile steht, was Intuition zu bearbeiten hat, nämlich Window__Gadgets und Extras. Ich habe das ein paar Zeilen höher als Equates hingeschrieben. Beide Equates hätten auch im »dc.l« stehen können, nur wäre dann das Listing zu breit geworden. In der ersten Equate-Zeile stehen die Window-Gadgets, die installiert werden sollen. Die symbolischen Namen (aus dem Include-File) sind wohl eindeutig.

In der nächsten Zeile habe ich dem Window noch zwei Eigenschaften verpaßt. Smart Refresh heißt, daß der betroffene Inhalt des Fensters gerettet werden soll, wenn es von einem anderen ganz oder teilweise überdeckt wird. Sobald das Fenster wieder oben liegt oder der abgedeckte Teil wieder sichtbar wird, zeichnet Intuition den Fensterinhalt neu. Im Gegensatz dazu gibt es noch Simple Refresh. In diesem Fall meldet Intuition nur den »Schaden«.

Nach dem Öffnen des Fensters kann man im Fenster zeichnen. Mit Move() wird die Zeichenposition gesetzt, mit Text() ab dieser Stelle ein String ausgegeben. In beiden Fällen muß man den RastPort wissen. Graphics benötigt immer die Adresse des RastPorts, dem ein Window zugeordnet ist. Ein RastPort ist vereinfacht ausgedrückt eine Struktur, die die Zeichenbedingungen etwas ausführlicher beschreibt, als ein Fenster. Seine Adresse kann man sich mit

```
move.l  wd_RPort(A1),A1
```

holen, wenn vorher der Window-Zeiger in A1 geladen wurde. Dahinter steckt immer eine Technik, die ich Ihnen an folgenden Zeilen (Auszug aus dem Listing) verdeutlichen möchte:

```
move.l  windowptr,a0
;zeige auf Window-
Struktur
move.l  wd_UserPort(a0),
a0 ;nun auf Message-Port
move.b  MP_SIGBIT(a0),d1
;Anzahl Signal Bits ->
  d1
```

Unsere Window-Struktur im Listing ist nur eine Hilfskonstruktion. Nach dem Aufruf von OpenWindow erhalten wir in D0 einen Zeiger auf eine ähnliche Struktur, die Intuition für uns anlegt. Unsere Struktur können wir danach wegwerfen, ändern, den Speicherbereich anders belegen oder sie modifiziert für ein weiteres Fenster benutzen.

Wichtig ist, daß wir uns den Zeiger gut merken, hier in der Variablen »windowptr«. Die erste der drei Zeilen ist noch einfach: Der Zeiger wird in das Register A0 geladen. Die Konstante »wd_UserPort« ist das Offset vom Beginn der Window-Struktur auf ein Langwort innerhalb der Struktur. Dieses Langwort ist aber selbst auch nur ein Zeiger auf eine andere Struktur, nämlich den Window-User-Port. Innerhalb dieser Struktur gibt es ein Byte, in dem das Message-Port-Signal-Bit notiert ist. Genau das brauchen wir.

Auf diese Art greift man also auf einen der beiden IDCM-Ports zu. Intuition richtet automatisch zwei dieser Ports zu jedem Window ein. Der Empfangs-Port heißt UserPort, senden kann man über den WindowPort. In diesem Beispiel zwar überflüssig, aber um es mal zu zeigen: Benutzt man Wait(), muß man sagen, bei welchen Signal-Bits (praktisch Semaphoren) man geweckt werden möchte. Da zu jedem Port ein Signal-Bit gehört, müssen wir natürlich feststellen, welches unseres ist. Im Feld MP_SIGBIT steht, welches Bit das ist (als Bit-Nummer), die Funktion Wait() erwartet aber eine Maske in D0, in der genau dieses Bit gesetzt ist. Deshalb lade ich das Register D1 mit der Bit-Nummer, lade dann D0 mit 1 und schiebe nun mit LSL diese Eins auf den richtigen Platz. Sobald das Bit »klingelt«, wurde unser Window angesprochen.

Da wir als Event nur CLOSEWINDOW zugelassen hatten, können wir uns weitere Tests sparen und mit dem Schließen des Windows sowie aller offenen Libs das Programm beenden.

Fast alle Programme, die wir bisher geschrieben haben, konnten wir nur mit ihren Namen im CLI aufrufen. Das ist nicht die schlechteste Lösung, denn viele CLI-Kommandos sind auch nur Programme dieser Art, doch Sie wissen selbst, daß der Amiga mehr kann.

Programm-Betriebsarten

Sie wissen auch, daß jedes CLI (mit NEWCLI können Sie zusätzliche CLI's schaffen) ein Task ist. Unsere Programme waren für den CLI-Task, von dem aus sie aufgerufen wurden, praktisch nur Unterprogramme. Solange unser Programm lief, also zum Beispiel auf eine Eingabe wartete, war auch das CLI in diesem Unterprogramm und wartete.

Die nächsthöhere Stufe ist ein Programm, das im CLI mit »RUN Name« aufgerufen werden kann. Dieses Programm läuft dann wirklich als eigener Task, und das CLI ist wieder frei für andere Dinge.

Die höchste Stufe bilden dann Programme, die man von der Workbench aus starten kann, indem man einfach mit der Maus auf ihr Icon klickt. Ziel dieses Kapitels ist es, ein solches Programm zu erstellen, und natürlich zu zeigen, wie man dazu vorgehen muß.

Bei der Gelegenheit sollen gleich zwei Fliegen mit einer Klappe erschlagen werden. Es gibt da noch eine Art von Programmen, nämlich solche, die sowohl vom CLI als auch von der Workbench aus gestartet werden können. Ich glaube, wir sind uns einig, daß jedes Workbench-Programm auch unter dem CLI laufen sollte, folglich können wir auf die Lösung »Nur Workbench« getrost verzichten. Wie Sie gleich sehen werden, ist die damit erzielbare Einsparung auch nur minimal.

Fassen wir zusammen. Es gibt:
1. CLI-Unterprogramme (Aufruf mit Namen),
2. CLI-Tasks (Aufruf mit RUN Namen),
3. Workbench-Tasks (Klick auf das Icon),
4. Kombination von 2. und 3.

Die Gruppen 1 und 2 unterscheiden sich nur minimal. Gruppe 1 benutzt für die Eingabe und Ausgabe das CLI-Window. Das Handle dafür wird mit der DOS-Funktion Input beziehungsweise Output ermittelt.

Gruppe 2 arbeitet nicht mit diesen Handles, sondern mit einem eigenen Fenster. Die Programme im Abschnitt »Verzweigungen und Menütechnik« (Listing 10, 11, 12, 13) gehören dazu. Probieren Sie es aus und starten diese Programme mit »RUN Name«. Achten Sie aber bitte darauf, daß Sie immer erst mit der Maus das Fenster anklicken müssen, in dem Sie arbeiten wollen. Wenn Sie genau hinsehen, werden Sie feststellen, daß auf dem Schirm nach dem Aufruf auch »CLI2« steht. Praktisch heißt das: CLI 1 richtet für das Programm temporär (solange es läuft) ein neues CLI ein.

Workbench-Programme (Gruppe 3) müssen eine Zusatzbedingung erfüllen. Sie dürfen nicht einfach loslaufen, wann Sie wollen, sondern müssen sozusagen auf die Starterlaubnis der Workbench warten. Deshalb müssen solche Programme zu Beginn auf eine Message (den Startbefehl) warten. Wenn Sie fertig sind, müssen Sie das der Workbench melden, indem Sie genau diese Message (die Sie sich gut gemerkt haben) an die Workbench zurückschicken.

Wird dasselbe Programm vom CLI aus aufgerufen, entfällt natürlich diese Geschichte. Das heißt auch, daß unser Programm unterscheiden muß, von wo es aufgerufen wurde.

Der Trick hinter der ganzen Geschichte ist der sogenannte Startup-Code. Der Name ist nicht ganz korrekt, hat sich jedoch so eingebürgert. Zum Start-Code gehört nämlich immer auch ein Ende-Code. Um nun beides in einen einzigen File packen zu können, den man bei HiSoft und SEKA als Include-File zuerst lädt und bei

```
* startup.i

* Startup-Code fuer Assembler-Programme. Recht frei nach dem
* Beispiel im ROM-Kernel-Manual Libraries and Devices, aber so
* geschrieben, dass als Include-File brauchbar und auf das
* wirklich notwendige reduziert
* -----------------------------------------------------------

        incdir  ":include/"

        include "exec/exec_lib.i"
        include "libraries/dosextens.i

        movem.l d0/a0,-(sp)                 ;rette Kommandozeile
        clr.l   _WBenchMsg                  ;sicherheitshalber

* Teste, von wo wir gestartet wurden
* ----------------------------------
        sub.l   a1,a1                       ;a1=0 = eigener Task
        CALLEXEC        FindTask            ;wo sind wir?
        move.l  d0,a4                       ;Adresse retten

        tst.l   pr_CLI(a4)                  ;Laufen wir unter WB?
        beq.s   fromWorkbench               ;wenn so

* Wir wurden vom CLI gestartet
* ----------------------------
        movem.l (sp)+,d0/a0                 ;Parms Kommandozeile
                                             holen
        bra     run                         ;und starten

* Wir wurden von Workbench gestartet
* ----------------------------------
fromWorkbench
        lea         pr_MsgPort(a4),a0
        CALLEXEC    WaitPort                ;Warte auf Start-Message
        lea         pr_MsgPort(a4),a0       ;sie ist da
        CALLEXEC    GetMsg                  ;hole sie
        move.l      d0,_WBenchMsg           ;immer Msg sichern!

        movem.l     (sp)+,d0/a0             ;bringe Stack i.O.

run
        bsr.s       _main                   ;rufe unser Programm auf

        move.l      d0,-(sp)                ;rette seinen Return-Code

        tst.l       _WBenchMsg              ;gibt's eine WB-Message
        beq.s       _exit                   ;nein: dann war's CLI

        CALLEXEC    Forbid                  ;keine Unterbrechung

        move.l      _WBenchMsg(pc),a1       ;hole die Message
        CALLEXEC    ReplyMsg                ;und gib sie zurueck
_exit
        move.l      (sp)+,d0                ;hole Return-Code
        rts                                 ;das war's

_WBenchMsg
        ds.l        1
        cnop        0,2
```

Listing 17. Der sogenannte Startup-Code

Metacomco mit dem Linker vor sein Programm setzt, kommt wieder ein kleiner Trick zum Tragen. Normalerweise hieße die Folge:

```
Start-Code
Unser Programmteil
End-Code
```

Praktisch gehen wir aber so vor:

```
Start-Code
jsr _main
End-code
_main Unser Programmteil
rts
```

Sie wissen jetzt, warum ich in all meinen Listings die Label »_main« an den Anfang gesetzt habe (wenn Sie sie nicht sehen, _main steckt im Include-File OpenDOS.i). Bei Metacomco sollten Sie auf jeden Fall mit »_main« arbeiten, der Linker erwartet dies, wenn Sie »startup.o« einbinden. Auch dürfte jetzt klar sein, daß unsere Programme immer mit einem schlichten aber wichtigen »rts« enden müssen.

Doch nun zur Praxis. Listing 17 zeigt das Startup. Kern der Angelegenheit sind diese Zeilen:

```
sub.l     a1,a1
;a1=0 = eigener Task
CALLEXEC FindTask
wo sind wir?
move.l  d0,a4
;Adresse retten
tst.l   pr_CLI(a4)
;Laufen wir unter WB?
beq.s    fromWorkbench
;wenn so
```

Die Exec-Funktion »FindTask« findet die Adresse einer Task-Kontroll-Struktur. Normalerweise übergibt man der Funktion die Adresse eines Strings mit dem Task-Namen im Register a1. Ist dieser Zeiger Null, erhält man die Adresse des eigenen Tasks.

Nun muß ich leider gestehen, daß hier »Task« nicht korrekt ist. Wir laufen unter einem DOS-Prozeß. Ein Prozeß ist so etwas Ähnliches wie ein Task, nur höherwertiger. »FindTask« gibt deshalb die Adresse unseres PLB (Prozeß-Leit-Block) zurück. Wenn Sie sich diese Struktur ansehen möchten: sie steht im Include-File »include/libraries/dosextens.i«.

Innerhalb dieser Struktur gibt es den Eintrag mit dem Offset »pr_CLI«. Das soll heißen »Pointer (Zeiger) auf den Command Line Interpreter«. Dieser Zeiger ist Null, wenn wir unter der Workbench laufen.

Also geht es in diesem Fall zur Label »fromWorkbench« und da steht:

```
fromWorkbench
lea     pr_MsgPort(a4),a0
ALLEXEC WaitPort
;Warte auf Start-Message
lea      pr_MsgPort(a4),
a0;sie ist da
CALLEXEC GetMsg
;hole sie
move.l  d0,_WBenchMsg
;immer Msg sichern!
```

Im PLB beim Offset »pr_MsgPort« steht die Adresse, die Exec-Funktion »Waitport« sehen will. Dieser Aufruf läßt unseren Task warten, bis er an die Reihe kommt. Stellen Sie sich das so vor: Es gibt eine Liste aller laufenden Tasks. Exec sorgt dafür, daß einer nach dem anderen für eine gewisse Zeit an die Reihe kommt, denn praktisch kann ja immer nur ein Task laufen (wir haben nur einen 68000 im Amiga). Seinen Startbefehl erhält der TASK über diesen Message-Port, genau: nur die Nachricht, daß eine Message da ist. Deshalb muß man mit »GetMsg« diese Nachricht aus dem Port lesen. Sie steht danach im Register d0. Da wir diese Nachricht noch brauchen, sichern wir sie in der Variablen »_WBenchMsg«.

Wenn unser Task unter der Workbench läuft, und er »geweckt« wurde, sind die Zeilen ab dem Label »run« interessant.

Mit »bsr _main« wird nun endlich unser Programmteil aufgerufen. Danach erfolgen die Rückzugsgefechte, das, was ich in der Einleitung den Ende-Code genannt habe. Wir müssen uns bei der Workbench ordnungsgemäß abmelden, was an sich dadurch geschieht, daß wir mit »ReplyMsg« die ursprünglich beim Start erhaltene Message zurückgeben. Da auch andere Tasks theoretisch zur selben Zeit auf den PLB zugreifen können, könnte es sein, daß unsere Nachricht nicht ankommt oder schlimmer, das totale Chaos ausbricht.

In einem Multitaskingsystem, in dem verschiedene Tasks auf gemeinsame globale Variable zugreifen können, gibt es deshalb immer einen Mechanismus, der einem Task für eine gewisse Zeit das alleinige Zugriffsrecht sichert. Diese Funktion heißt beim Amiga Forbid (Verbiete mir Störungen). Genaugenommen ist diese Funktion recht gefährlich, denn sie schaltet das Multitasking aus. Es bleibt ausgeschaltet, solange der Task läuft oder bis er Wait (warte auf Nachricht) oder Permit aufruft. Da wir nach dem Forbid nur noch die Reply-Message zurückgeben und

```
        opt      1-                                    ;nicht linken!

* free_ram

        incdir   ":include/"

        include startup.i                              ;oder wie Sie ihn nennen

        include intuition/intuition.i
        include intuition/intuition_lib.i
        include exec/memory.i
        include graphics/graphics_lib.i
        include libraries/dos_lib.i

GRAFIC  macro
                move.l  windowptr,a1                   ;Adresse Window-Struktur
                move.l  wd_RPort(a1),a1                ;von da auf Rast Port
                CALLGRAF        \1                     ;Grafik-Funktion
                endm

_main

* DOS-Library oeffnen
* -------------------
        lea             dosname,a1
        moveq           #0,d0
        CALLEXEC        OpenLibrary
        tst.l           d0
        beq             abbruch
        move.l          d0,_DOSBase                    ;Basis-Zeiger sichern

* Intuition Library oeffnen:
* --------------------------
        lea             intname,a1
        moveq           #0,d0
        CALLEXEC        OpenLibrary
        tst.l           d0
        beq             closedos
        move.l          d0,_IntuitionBase              ;Basis-Zeiger sichern

* Graphics Library oeffnen
* ------------------------
        lea             grafname,a1
        moveq           #0,d0
        CALLEXEC        OpenLibrary
        tst.l           d0
        beq             closeint
        move.l          d0,_GfxBase                    ;Basis-Zeiger sichern


* Window oeffnen
* --------------
        lea             windowdef,a0                   ;zeige auf Window-
Struktur
        CALLINT         OpenWindow                     ;oeffne Window
```

dann enden, ist das »Forbidding« hier vertretbar (und notwendig sowieso).

Nach dem »bsr _main« hatten wir noch mit

```
move.l  d0,-(sp)
```

den Return-Code unseres Programms gesichert. Es ist unsere Sache, was wir da zurückgeben wollen. Üblich ist Null für keinen Fehler. Dieses »d0« müssen wir natürlich vor dem RTS wieder vom Stack holen. Genauso hatten wir gleich zu Anfang des Programms mit

```
movem.l  d0/a0,-(sp)
;rette Kommandozeile
clr.l     _WBenchMsg
;sicherheitshalber
```

die Länge und Adresse einer eventuellen Kommandozeile gesichert und die Variable _WBenchMsg auf null gesetzt. Das Aufräumen des Stacks mit

```
movem.l  (sp)+,d0/a0
```

erfolgt dann jeweils entweder im CLI- oder im Workbenchzweig.

Schauen wir uns zum Schluß noch an, was denn tatsächlich im Falle von CLI geschieht, so bleibt:

```
run          bsr.s    _main
rts
```

Wenn Sie diesen Quelltext nun assemblieren, müßte das mit einer Ausnahme ohne Fehler über die Bühne gehen. Die Ausnahme ist das fehlende Label »_main«. Nun speichern Sie diesen File bitte unter dem Namen »startup.i«

Um einmal zu zeigen, daß einer unserer Tasks tatsächlich ständig läuft und etwas tut, wol-

```
        tst.l       d0                  ;ging was schief?
        beq         closegraf           ;wenn ja
        move.l      d0,windowptr        ;Window-Zeiger sichern

* Hauptschleife
* -------------
loop
        moveq       #MEMF_PUBLIC,d1     ;freien RAM
        CALLEXEC    AvailMem            ;einlesen
        move.l      d0,d2               ;nach d2
        lea         buffer,a0           in Hex-String
        bsr         hex

        moveq       #80,d0              ;X-Position fuer Text
        moveq       #19,d1              ;Y
        GRAFIC      Move                ;Move TO X,Y

        lea         buffer,a0           ;Text-Adresse
        addq.l      #2,a0               ;die 2 ersten Nibble sind
        moveq       #6,d0               ;eh 0, restliche 6 reichen
        GRAFIC      Text                ;Text zeichnen

        move.l      windowptr,a0        ;Von unserem Window
        move.l      wd_UserPort(a0),a0  ;den Empfangsport
        CALLEXEC    GetMsg              ;testen
        tst.l       d0                  ;Haben wir Post?
        bne         fini                ;kann nur CLOSEWINDOW sein

        move.l      #25,d1              ;25/50 = 1/2 Sekunde
        CALLDOS     Delay               ;warten
        bra         loop                ;und von vorn

fini
        move.l      d0,a1               ;Message ist in d0
        CALLEXEC    ReplyMsg            ;antworten

closewindow
        move.l      windowptr,a0        ;Fenster zu
        CALLINT     CloseWindow

closegraf
        move.l      _GfxBase,a1         ;Die Libs schliessen:
        CALLEXEC    CloseLibrary

closeint
        move.l      _IntuitionBase,a1
        CALLEXEC    CloseLibrary

closedos
        move.l      _DOSBase,a1
        CALLEXEC    CloseLibrary

abbruch
        moveq       #0,d0               ;keinen Fehler melden
        rts                             ;und Ende

* Konvertiere d2.l in ASCII-String ab (a0)
* ----------------------------------------
hex
        moveq       #8-1,d1             ;nun alle Nibble
next
        rol.l       #4,d2               ;hole 1 Nibble
        move.l      d2,d3               ;nach d3 retten
        and.b       #$0f,d3             ;maskiere es
        add.b       #48,d3              ;in ASCII wandeln
        cmp.b       #58,d3              ;ist es >9 ?
        bcs         out                 ;wenn nicht
        addq.b      #7,d3               ;sonst muss es A-F sein
out
        move.b      d3,(a0)+            ;1 Zeichen abspeichern
        dbra        d1,next             ;next nibble
        rts


W_Gadgets       equ     WINDOWDRAG!WINDOWDEPTH!WINDOWCLOSE
W_Extras        equ     SMART_REFRESH!ACTIVATE

W_Title         dc.b    ' Freier RAM z.Z. ',0

windowdef
                dc.w    200,20          ;links, oben
                dc.w    220,24          ;Breite, Hoehe
                dc.b    -1,-1           ;Pens des Screen
                dc.l    CLOSEWINDOW     ;einiziges IDCMP Flag
                dc.l    W_Gadgets!W_Extras;Window Flags
                dc.l                    ;keine User-Gadgets
                dc.l    0               ;keine User-Checkmark
                dc.l    W_Title         ;Titel des Window
                dc.l                    ;kein eigener Screen
                dc.l    0               ;keine Super Bitmap
                dc.w    100,20          ;Min. Groesse
                dc.w    640,200         ;Max.
                dc.w    WBENCHSCREEN    ;Use Workbench Screen

intname         INTNAME                 ;Namen der Libs aus Makros
grafname        GRAFNAME
dosname         DOSNAME

buffer          ds.b    8

_IntuitionBase  ds.l    1               ;Speicher fuer die Zeiger
_GfxBase        ds.l    1
_DOSBase        ds.l    1
windowptr       ds.l    1
```

Listing 18. Dieses Programm zeigt immer den freien Speicherplatz

len wir nun ein Programm schreiben, das in einem Window ständig den noch freien RAM anzeigt (Listing 18).

Nachdem alle Libraries, die wir hier benötigen, geöffnet sind, kann es losgehen. Der Kern des Programms steckt in den fünf Zeilen ab »loop«. Wir holen uns den freien Speicher, indem wir der Exec-Funktion »AvailMem« die Konstante »MEMF_PUBLIC« übergeben. Diese Konstante ist im Include-File »include/exec/memory.i« mittels EQU definiert. Sie werden dort auch noch einige andere Parameter finden. Machen Sie sich einmal den Spaß, und ändern das Programm so, daß auch die Größen von Fast-RAM, Chip-RAM etc. angezeigt werden.

Nun folgt die Hex-Konvertierung, die Sie schon kennen, und die Ausgabe des Textes, wie im Beispiel von Listing 16. Neu ist nur der Makro GRAFIC, der mir einiges an Tipperei erspart. Auch anders ist das Warten auf ein Intuition-Event gelöst. Auch hier erwarten wir nur gemäß unserer Window-Definition nur eines, nämlich CLOSEWINDOW.

Wie Sie sehen, reicht es, mit »CALLEXEC GetMsg / tst.l d0« einfach den Message-Port zu lesen. Ist der »Briefkasten« leer, ist d0 null. Normaler ist diese Methode des sogenannten Pollings unschön, doch hier geht es, weil wir im Falle von keine Nachricht mit diesen Zeilen weitermachen:

```
move.l  #25,d1
;25/50 = 1/2 Sekunde
CALLDOS Delay    ;warten
bra     loop
;und von vorn
```

Mit dem Aufruf der Delay-Routine geben wir nämlich den anderen Tasks für eine halbe Sekunde Zeit (aus CPU-Sicht die reinste Ewigkeit) selbst etwas zu tun. Wenn alle halbe Sekunde der neue Stand des RAM angezeigt wird, dürfte es wohl reichen. Wer schneller informiert sein will, kann die Zeit natürlich kürzer wählen.

Wurde das Close-Gadget des Fensters angeklickt, dann haben wir eine Message erhalten und es erfolgt ein Sprung zum Label »fini«. Dort steht:

```
fini    move.l  d0,a1
;Message ist in d0
CALLEXEC ReplyMsg
;antworten
```

und das möchte ich Ihnen noch einmal besonders ans Herz legen. Wir müssen Intuition auf jede Nachricht antworten! Das geschieht ganz einfach durch den Vermerk »zurück an Absender«, sprich wir senden die eben erhaltene Nachricht zurück.

Ist das Programm (fehlerfrei) assembliert und heißt das Ergebnis »free__ram«, dann können Sie jetzt ins CLI gehen und »run free__ram« tippen. Nun sollte das Fenster mit der Anzeige erscheinen. Um nun ein CLI-Kommando ausführen zu können, müssen Sie zuerst irgendwo im CLI-Fenster klicken. Geben Sie nun DIR ein, und Sie werden sehen, wie sich die Speicheranzeige ständig ändert, solange DIR läuft.

Einen kleinen Fehler hat unser Programm noch. Wir wollen ja in sein Fenster nichts eingeben, warum also ist das Fenster

aktiv, und warum müssen wir erst das CLI-Fenster anklicken? Die Lösung ist recht einfach. Ändern Sie die Zeile

```
W_Extras        equ
SMART_REFRESH!ACTIVATE
```

in

```
W_Extras        equ
SMART_REFRESH
```

Damit ist das Fenster nicht mehr aktiv, sein Titel wird dann in Geisterschrift dargestellt. Sie können es natürlich anklicken, wenn Ihnen die aktive Form besser gefällt.

Um ein Programm, das mit einem ordentlichen Startup-Code versehen ist, von der Workbench aus starten zu können, benötigt es nur noch ein Icon. Damit ein Icon sichtbar wird, muß es erstens vorhanden sein (logisch) und zweitens in einem Directory stehen, das selbst ein Icon hat. Am einfachsten stellen Sie eine solche Schublade her, indem Sie auf der Workbench die Schublade »Empty« duplizieren. Sehen Sie deren Icon nicht, schließen Sie das Disk-Fenster und öffnen es wieder. Nun ziehen Sie die Empty-Schublade etwas weg, und die Test-Schublade ist da.

Jetzt brauchen wir ein Programm-Icon. Dazu nehmen Sie am besten auch ein vorhandenes Icon, allerdings nicht jedes ist geeignet. Der Amiga kennt verschiedene Typen von Icons. Welche das sind und was sie für eine Bedeutung haben, erfahren Sie automatisch, wenn Sie den Icon-Editor starten. Für uns ist es wichtig, zu wissen, daß Programme vom Typ TOOL sein müssen.

Geeignet ist zum Beispiel IconED selbst. Nehmen wir an, Sie haben das Directory (die Schublade) »test« schon erstellt und unser Programm hieße »free_ram«. Dann kopieren Sie zuerst das Programm mit

```
copy free_ram :
test/free_ram
```

Nun kopieren Sie ein Icon dazu (IconEd steckt im System-Ordner):

```
copy :system/iconed.
info :test/free_ram.info
```

Nun sollten Sie auf der Workbench in der Schublade »test« ein Icon finden, das aussieht, wie das von IconEd, aber den Titel »free_ram« zeigt. Dieses können Sie nun getrost anklicken, »free_ram« wird anschließend starten.

Wenn Sie jetzt Ihrem Icon ein eigenes Aussehen verpassen wollen, rufen Sie IconEd auf. Im Disk-Menü wählen Sie LOAD und tippen dann in den Text-Requester

```
:test/free_ram
```

(immer den vollen Pfadnamen).

Das Editieren ist im Prinzip selbsterklärend. Probieren Sie einfach die verschiedenen Menü-Punkte aus.

Um ein Icon zeichnen/ändern zu können, müssen Sie immer das Menü Color anwählen und daraus die passende Farbe. Radieren können Sie mit der Hintergrundfarbe. Gezeichnet/radiert wird mit der Maus. Die linke Taste drückt den Stift auf das »Papier«. Sie können auch auf den Kopiervorgang ganz verzichten und ein Icon selbst im Editor erstellen. Sie müssen dann nur im Save-Requester den korrekten Namen eingeben:

```
test/free_ram
```

Ansonsten keine Sorge. Es können zwar die unmöglichsten Icons entstehen, aber editiert wird immer nur der Info-File. Ihrem Programm passiert nichts.

Fast zum Schluß, aber noch rechtzeitig, müssen wir noch einmal auf den 68000 zurückkommen, aus dessen Befehlssatz wir bisher nur einen Teil behandelt haben.

In den 68000-Datenbüchern finden Sie zu jedem einzelnen Befehl die Syntaxformen, die erlaubten Adressierungsarten, die Längen, die Laufzeiten und vieles mehr. So ein Buch mit 300 Seiten können wir hier leider nicht abdrucken. Trotzdem wollen wir Ihnen einen Überblick geben. Meistens kommen Sie damit und der Regel »Befehl Quelle,Ziel« schon ganz gut zurecht. Zum anderen finden Sie im AMIGA-Magazin und im AMIGA-Sonderheft Listings, die zeigen, wie man die Befehle anwendet. Auf den folgenden Seiten können Sie dann nachlesen, was sie bedeuten.

Am häufigsten werden Sie die Transfer-Befehle laut Tabelle 4 einsetzen. Auffallend sind sicherlich die vielen Varianten des MOVE-Befehls. Hier sollten Sie einmal Ihren Assembler testen. Gute Assembler akzeptieren auch ein MOVE, wo man eigentlich MOVEA hätte schreiben müssen. Sehr gute Assembler geben sogenannte Warnings aus, wenn Sie nicht optimal programmieren, hier also anstatt eines zulässigen MOVEQ nur ein einfaches MOVE schreiben würden. Besonders erwähnenswert sind die LINK und UNLK (Unlink). Mit diesen Befehlen ist der 68000 besonders gut auf die Aufgabenstellung von Hochsprachen-Compilern vorbereitet. Hier ergibt sich immer das Problem, daß in Prozeduren und Funktionen lokale Variable geschaffen werden müssen, die nur so lange existieren sollen, wie die Prozedur (Funktion) aktiv ist. Typisch werden deshalb solche Variablen auf dem Stack abgelegt. Ideal ist es nun, wenn man mit nur einem Befehl den passenden Stackbereich reservieren und dann später auf genauso einfache Art wieder freigeben kann. Genau das bieten die Befehle LINK und UNLK. Wenn nun eine Prozedur (Unterprogramm) eine weitere Prozedur aufruft und diese dann eine dritte etc., und jede dieser Prozeduren mit dem eigenen lokalen Stack arbeitet, dann entsteht sozusagen eine verkettete Liste, englisch »linked list«. Daher rühren auch die Namen LINK und UNLK.

Schauen wir uns jetzt einmal an, wie das funktioniert. Die Syntax des Befehls lautet:

```
LINK An,#Adreßdistanz
```

Ein Beispiel:

```
LINK A6,#30
```

In diesem Fall wird zuerst A6 auf dem Stack abgelegt, praktisch der Befehl:

```
move.l  a6,-(sp)
(Schritt 1)
```

ausgeführt. Nun wird der Stackpointer in das soeben gerettete Register kopiert:

```
move.l  sp,a6
(Schritt 2)
```

Zuletzt wird die Adreßdistanz auf den Stackpointer addiert:

```
add.l   #Adr_Dist,sp
```

Die Transfer-Befehle des 68000

Befehl	Bedeutung
EXG	Austausch von Registerinhalten
LEA	Laden eines Registers
LINK	Lokalen Stack aufbauen
MOVE	Übertragen (Kopieren) von Daten
MOVEA	Übertragen (Kopieren) von Adressen
MOVEM	Übertragen (Kopieren) mehrerer Register
MOVEP	Übertragen (Kopieren) von Daten zur Peripherie
MOVEQ	Übertragen (Kopieren) von Konstanten »Quick«
PEA	Adresse auf den Stack bringen
SWAP	Vertauschen der Worte eines Registers
UNLK	Abbau des lokalen Stacks (siehe LINK)

Tabelle 4. Die Transfer-Befehle des 68000

Arithmetik-Befehle des 68000

Befehl	Bedeutung
ADD	Addition von Daten
ADDA	Addition von Adressen
ADDI	Addition einer Konstanten
ADDQ	Addition einer Konstanten »Quick«
ADDX	Addition mit Übertrag-Bit
CLR	Löschen eines Operanden
CMP	Vergleich zweier Daten
CMPA	Vergleich zweier Adressen
CMPI	Vergleich mit einer Konstanten
CMPM	Vergleich zweier Daten im Speicher
DIVS	Division mit Vorzeichen
DIVU	Division ohne Vorzeichen
EXT	Vorzeichenrichtige Erweiterung
MULS	Multiplikation mit Vorzeichen
MULU	Multiplikation ohne Vorzeichen
NEG	Negation
NEGX	Negation mit X-Bit
SUB	Subtraktion von Daten
SUBA	Subtraktion von Adressen
SUBI	Subtraktion einer Konstanten
SUBQ	Subtraktion einer Konstanten »Quick«
SUBX	Subtraktion mit X-Bit (Borgen)
TST	Teste Operanden gegen Null
ABCD	Addition von BCD-Zahlen
NBCD	Negation von BCD-Zahlen
SBCD	Subtraktion von BCD-Zahlen

Tabelle 5. Mächtige Arithmetik-Befehle bietet der 68000

Das Prinzip der BCD-Arithmetik

Zahl	Wert	im Speicher auf Adressen
1	123456	12 auf 1001, 34 auf 1002, 45 auf 1003
2	654321	65 auf 2001, 43 auf 2002, 21 auf 2003

Tabelle 6. Das Prinzip der BCD-Arithmetik

```
move     #1004,a1
move     #2004,a2
move     #4,CCR
ABCD     -(a1),-(a2)
ABCD     -(a1),-(a2)
ABCD     -(a1),-(a2)
```

Listing 19. Das Listing zur Tabelle 5

Damit hätten wir den lokalen Stack für ein Unterprogramm. Üblicherweise wählt man die Adreßdistanz negativ, da der Stack zu fallenden Adressen hinwächst. Mit UNLK wird der ursprüngliche Zustand wiederhergestellt. Praktisch wirkt UNLK wie

```
move.l  a6,a7
move.l  (sp)+,a6
```

Für das Hauptprogramm oder allgemein das aufrufende Unterprogramm hat somit das Adreßregister und der Stackpointer wieder seinen ursprünglichen Wert.

Zu beachten ist noch, daß das Adreßregister nach dem LINK-Befehl eine Kopie des Stackpointers hält. Somit kann das Unterprogramm sehr einfach auf Daten des aufrufenden Programms zugreifen, wenn dieses die Daten vorher auf den Stack gepackt hat.

Der 68000 bietet laut Tabelle 5 eine Menge sehr leistungsfähiger Arithmetik-Befehle, die Sie aber nicht unbedingt alle beherrschen müssen. Auch hier können gute Assembler Ihnen einige Arbeit abnehmen. Zumindest sollten Sie mit ADD einverstanden sein, wenn eigentlich ADDA oder ADDI erforderlich ist. Sinngemäßes gilt für CMP und SUB.

Die arithmetischen Operationen sind auf einer Breite von 32 Bit möglich. In diesem Sinne ist also der 68000 ein echter »32-Bitter«. Sein Datenbus ist zwar nur 16 Bit breit, was dazu führt, daß Langworte in »zwei Portionen« transportiert werden, das interessiert aber nur sekundär. Primär ist wichtig, daß damit mathematische Operationen wesentlich einfacher zu programmieren sind, als auf CPUs, die nur 16 oder gar nur 8 Bit breite Operanden zulassen. Sekundär sollte man natürlich darauf achten, daß die Daten solange als möglich in Registern gehalten werden. Dann entfällt auch der relativ zeitaufwendige Transfer über den Datenbus.

Logik-Befehle

Befehl	Bedeutung
AND	Logisch UND
ANDI	Logisch UND mit einer Konstanten
EOR	Logisch XOR
EORI	Logisch XOR mit einer Konstanten
NOT	Logisch NICHT (Einerkomplement)
OR	Logisch ODER
ORI	Logisch ODER mit einer Konstanten

Tabelle 7. Die logischen Befehle in der 68000er-Syntax

Bit-Operationen

Befehl	Bedeutung
BCHG	Ändere (kippe) ein Bit
BCLR	Lösche ein Bit
BSET	Setze ein Bit
BTST	Prüfe ein Bit
TAS	Teste und setze Bit 7 eines Byte-Operanden

Tabelle 8. Das »Bit-Schieben« wird kräftig unterstützt

Bei der hohen Geschwindigkeit des 68000 sollte man das allerdings nicht überbewerten. Nur in sehr rechenintensiven Routinen könnte damit etwas erreicht werden.

Die BCD-Arithmetik des 68000 wird jeder schätzen, der schon einmal auf einer anderen CPU so etwas programmieren mußte. Eine BCD-Ziffer steht immer in einem Halbbyte (4 Bits). Da sich damit die Zahlen 0 bis 15 darstellen lassen, hier aber nur 0 bis 9 gültig sind, gibt es beim Überlauf einige Probleme. Bei anderen CPUs muß man diesen Fall mit Hilfe des sogenannten Half-Carry-Flags testen, hier kann man einfach addieren. Da als Operandengröße immer nur Byte zugelassen ist, gibt es den Überlauf nach 99. Dieser geht aber automatisch in das X-Flag und wird auch immer mitaddiert.

Tabelle 6 zeigt den Speicherauszug und das Programm für die Addition zweier sechsstelliger Zahlen in je 3 Bytes. Wie üblich muß man rechts (bei den Einern) mit der Addition beginnen. Daher ist als Adressierungsart hier nur »ARI mit Predekrement« erlaubt.

Dabei sind drei Dinge zu beachten:

1. Wegen des Predekrements muß das Byte mit dem »Einer« auf einer ungeraden Adresse liegen.
2. Um nicht beim ersten Mal ein zufälliges X-Flag mitzuaddieren, muß man es löschen.
3. Um ein Null-Ergebnis erkennen zu können, sollte man vorher das Z-Flag setzen.

Die Punkte 2 und 3 lassen sich mit der Anweisung »move #4,CCR« sehr einfach zusammen erledigen. Eines bleibt Ihnen allerdings nicht erspart. Sie müssen schon garantieren, daß die Zahlen BCD-Zahlen sind (Abfrage beim Laden). Größere Werte als 9 werden nämlich schlicht falsch addiert.

Der 68000 beherrscht die Logik-Befehle von Amiga-Basic? Nein, tatsächlich ist es umgekehrt. Auf jeden Fall brauchen wir hier nicht viel darüber zu schreiben, denn die Logik-Funktionen sind sehr schön im Amiga-Basic-Handbuch erklärt. Tabelle 7 zeigt also nur die 68000-Syntax.

In Assembler ist man häufig mit dem »Bit-Schieben« beschäftigt. Der 68000 unterstützt uns dabei mit den Befehlen, die uns Tabelle 8 zeigt.

Sie müssen nur beachten, daß die direkte Bit-Angabe wie »BTST #3,D5« nur bis Bit 7 erlaubt ist. Größere Werte (8 bis 31) müssen Sie vorab in ein Datenregister laden, also zum Beispiel für Bit 13 schreiben:

```
move #13,d0
btst d0,d5
```

Die Bit-Befehle halten immer den vorherigen Zustand im Z-Flag fest, führen also zwei Operationen aus. Diese Eigenschaft, und ganz besonders die Fähigkeit von TAS sind ein gutes Beispiel für eine besondere Fähigkeit des 68000, nämlich die Unterstützung von Multi-Tasking.

Mit dem TAS-Befehl wird Bit 7 eines Byte-Operanden im Speicher abgefragt und das Ergebnis wie üblich im Z-Flag notiert. Dann wird eine 1 in Bit 7 geschrieben. Die ganze Folge, also Lesen des Operanden, Abfrage und Zurückschreiben ist unteilbar, sprich kann nicht durch einen Interrupt unterbrochen werden. Solche unteilbaren Befehle sind in einem Multi-Tasking-System sehr wichtig. Hier wären zum Beispiel Aufgaben wie Prozeßumschaltung, Synchronisation von Prozessen und ähnliche zu nennen.

Schiebe- und Rotierbefehle sind auch typisch für Assemblerprogramme. Einige hatten wir schon kennengelernt, Tabelle 9 zeigt den Gesamtüberblick:

Anzumerken ist auch hier, daß wie bei den Bit-Befehlen direkt nur #0 bis #7 erlaubt sind, und dann wieder ein Datenregister dazugenommen werden muß.

Neben den Transfer-Befehlen werden Sie die Programmsteuer-Befehle am häufigsten verwenden. Tabelle 10 zeigt, welche Möglichkeiten der 68000 bietet.

Zu beachten ist hier der Unterschied zwischen den Verzweigungs- und den Sprungbefehlen. Erstere sind immer relativ zum aktuellen PC, allerdings auf einen Adreßbereich von ±32 KByte begrenzt.

Die Sprungbefehle JMP und JSR reichen über den vollen Adreßbereich von 16 MByte.

CHK testet ein Datenregister gegen zwei Grenzen, nämlich 0 und eine im Operanden angegebene Grenze. Spricht der Test an, wird eine Exception ausgelöst. Damit läßt sich sehr einfach eine Bereichsprüfung, zum Beispiel von Array-Indizes, realisieren.

Große Teile der Amiga-Dokumentation sind in C geschrieben, auch Intuition. Grund ge-

nug, sich um die Übersetzung in Assembler zu kümmern:

Ein »#include« ersetzen Sie durch »include«, beim File-Namen ändern Sie die Extension von h in i. Für

```
#define  Name  Wert
```

schreiben Sie

```
Name       equ   Wert
```

Die Namen der Funktionen stimmen überein, nur daß wir immer _LVO davorsetzen müssen. Nehmen wir allerdings die Makros, kommen wir C noch näher, denn zum Beispiel:

```
move.l_IntuitionBase,a6
jsr    _LVOOpenWindow(a6)
```

heißt mittels Makro nur

```
CALLINT OpenWindow
```

Die Datentypen in C machen viel Lärm um wenig, Tabelle 11 zeigt Typen im Vergleich:

Die groß geschriebenen Typen sind übrigens keine echten C-Typen, sondern nur Makros. Finden Sie in C eine Struct-Konstruktion der Form

```
struct Name        {
        INT     anton;
        LONG    berta;
        }
```

können Sie das umsetzen in:

```
anton   ds.w    1
berta   ds.l    1
```

Die Amiga-Strukturen (wie Window oder Screen) werden Sie schon anhand der Namen wiedererkennen. Manchmal finden Sie auch nur deren Namen mit dem Präfix »extern«. Das ist dann eine Anweisung an den Linker, diese Struktur einzubinden. In diesem Fall schauen Sie in den Include-Files des Devpac nach, wo alle Strukturen definiert sind.

Manchmal sind die Namen mit einem Präfix versehen, um Namenskonflikte zu vermeiden (In C dürfen Strukturen und Variablen denselben Namen haben, in Assembler nicht).

Das in C beim Aufruf übliche

```
if OpenWindow
(&windowptr)==0
   exit(false)
```

ersetzen Sie durch

```
tst.l   d0
beq             exit
;dort Close Libs und so
und rts
```

Ein Ausdruck wie »window prt-→wd_RPort« kostet uns in Assembler die Zeilen

```
move.l  windowptr,a1
move.l  wd_RPort(a1),a1
```

Damit wären wir am Ende unseres Kurses und Sie am Anfang Ihrer (hoffentlich steilen) Karriere als Assembler-Programmierer angekommen. Wenn Sie tiefer einsteigen wollen, sehen Sie sich die Literaturvorschläge unter der Rubrik »Bücher« in diesem Heft an (ab Seite 147).

Wir wünschen Ihnen viel Erfolg mit der maschinennahen Programmierung.

(Peter Wollschlaeger/rs)

Schiebe- und Rotierbefehle

Befehl	Bedeutung
ASL	Arithmetische Verschiebung nach links
ASR	Arithmetische Verschiebung nach rechts
LSL	Logische Verschiebung nach links
LSR	Logische Verschiebung nach rechts
ROL	Rotieren links herum
ROR	Rotieren rechts herum
ROXL	Rotieren mit X-Bit links
ROXR	Rotieren mit X-Bits rechts

Tabelle 9. Mit diesen Befehlen kommt der 68000 ins Rotieren

Befehle zur Programmsteuerung

Befehl	Bedeutung
Bcc	Verzweige bedingt
BRA	Verzweige immer
BSR	Verzweige zu einem Unterprogramm
CHK	Checke Datenregister gegen Grenze 0 und andere
DBcc	Bedingte Schleife
JMP	Sprung zu einer Adresse
JSR	Sprung zu einem Unterprogramm
NOP	Keine Operation
RESET	Rücksetzen der Peripherie
RTE	Rückkehr von einer Exception
RTR	Rückkehr mit Laden der Flags
RTS	Rückkehr aus einem Unterprogramm
Scc	Setze 1 Byte bedingt
STOP	Halte Programm an
TRAP	Gehe in Exception
TRAPV	Gehe in Exception, wenn V-Flag gesetzt

Tabelle 10. Alle Befehle zur Programmsteuerung

Datentypen

C-Typ	Assembler-Typ
int	.W
long int	.L
unsigned int	.W
char	.B
BYTE	.B
UBYTE	.B
WORD	.W
UWORD	.W
LONG	.L
ULONG	.L

Tabelle 11. Diese Datentypen in C entsprechen denen in Assembler

AMICUS
DIE ABENTEUER EINER FLOPPY-DISC







Fensterln mit PowerWindows

Beeindruckende Fenster, Menüs und Gadgets geben Amiga-Programmen den notwendigen Pfiff. Die Programmierung mit »Intuition« erfordert viel Geduld und führt häufig zu Abstürzen. »PowerWindows« bietet eine bequeme und sichere Hilfe, diese Hindernisse zu überwinden. Mit diesem Vorgang verwenden Sie die Steighilfe zum Fensterln sinnvoll und effektiv.

Was fällt einem Amiga-Programmierer zum Thema »Intuition« ein? Strukturen, Strukturen, Strukturen. Bevor er von der komfortablen Amiga-Benutzeroberfläche profitieren kann, muß der Programmierer erst einmal eine Unmenge von Intuition-Datenstrukturen mit den richtigen Werten versehen – eine fehlerträchtige und zeitraubende Angelegenheit. Doch mit »PowerWindows« wird alles anders: Definieren Sie Ihre Windows, Screens, Menüs, Gadgets und Requester mit der Maus und lassen Sie den Amiga die Schreibarbeit erledigen – wozu hat man schließlich den Computer?

PowerWindows ist schon seit einiger Zeit auf dem Markt. Seit der Version 2.5 arbeitet es mit den meisten verbreiteten Programmiersprachen zusammen. Leider steht die Anleitung dieses fantastischen Programms in keinem Verhältnis zu seiner Leistungsfähigkeit – sie ist mit mageren elf Seiten nur für Amiga-Profis verständlich. Doch PowerWindows kann gerade Anfängern eine Menge Arbeit abnehmen. Wir zeigen, wie Sie – egal ob Anfänger, Fortgeschrittener oder Profi – sich auf Ihr eigentliches Programm konzentrieren können und sich nicht mit den Strukturdefinitionen herumschlagen müssen.

Sie denken – der Amiga schreibt

Was ist dieses PowerWindows eigentlich? PowerWindows ist ein Programm, bei dem Windows, Screens, Menüs, Gadgets und Requester komfortabel mit Hilfe der Maus auf dem Bildschirm definiert werden. Die verschiedenen Intuition-Flags (dazu lesen Sie später mehr) werden einfach per Mausklick eingestellt. Menüs und Untermenüs werden von PowerWindows komfortabel verwaltet. Man kann dazu Menüpunkte einfach eingeben, ohne sich um irgendwelche Bildschirmkoordinaten kümmern zu müssen. Auch alle Formen von Gadgets werden einfach mit der Maus auf den Bildschirm gebracht, Menüs und Gadgets sind auf Wunsch mit selbst gezeichneten Grafiken zu veredeln. Obwohl das Programm sehr einfach zu bedienen ist, lassen sich (fast) alle Fähigkeiten von Intuition anwenden.

Hat man seine Definitionen mit der Maus beendet, ist PowerWindows dran: Es schreibt nun sämtliche benötigten Intuition-Datenstrukturen in Aztec C, Lattice C oder Assembler in eine Datei. Diese Datenstrukturen werden später einfach mit »#include« in eigene Programme eingebaut.

Der große Vorteil bei der Arbeit mit PowerWindows ist, daß man Ergebnisse und Änderungen direkt auf dem Bildschirm sieht, Fehler sofort erkennt. Selbst definierte Menüs lassen sich beispielsweise ganz normal als Amiga-Menüs anklicken und ausprobieren. Somit wird die Zeit für die Entwicklung von Amiga-Programmen erheblich verringert – und man kann sich endlich auf das Wesentliche konzentrieren.

Um bei unserem Workshop mitzumachen, sollten Sie zumindest etwas Programmiererfahrung in C oder Assembler mitbringen. Alle Beispielprogramme sind in C verfaßt; ein Assembler-Programmierer mit etwas Erfahrung sollte keine großen Schwierigkeiten haben, sie umzusetzen. Obwohl die Beispielprogramme kommentiert sind, können wir für C-Anfänger nicht alles genau erklären. In diesem Heft befindet sich auf Seite 24 ein C-Kurs, der Ihnen die Grundbegriffe der Betriebssystem-Programmierung in C vermittelt.

Bevor Sie loslegen, benötigen Sie natürlich das Programm PowerWindows. Dieser Kurs basiert auf der Version 2.0. Die soeben erschienene Version 2.5 unterstützt wesentlich mehr Programmiersprachen als die Version 2.0 und arbeitet mit der PAL-Auflösung des Amiga. Außerdem wurden einige kleine Fehler beseitigt. Wenn Sie die Version 2.0 besitzen, sollten Sie sich möglichst bald die Version 2.5 besorgen. Der Hersteller (Inovatronics) bietet einen Update-Service gegen geringe Gebühr und Einsendung des Originals.

Wenn Sie in C programmieren, benötigen Sie einen C-Compiler; wir haben »Aztec C« verwendet. Die Beispielprogramme dürften aber auch mit Lattice C keine Schwierigkeiten bereiten. Assembler-Programmierer brauchen einen Assembler, der Include-Dateien verarbeiten kann, beispielsweise »Devpac«, »Profimat« oder »CAPE«; daneben ist der »A68k«-Assembler von der Fish-Disk 110 eine interessante Alternative für alle, die wenig Geld investieren wollen. Der weitverbreitete »Seka«-Assembler ist leider für PowerWindows ungeeignet, da er die Amiga-Standard-Include-Dateien nicht verarbeiten kann. Alle Beispielprogramme in diesem Kurs sind in C geschrieben. Das erste Programm ist auch in Assembler abgedruckt, damit man sieht, wie PowerWindows-Dateien in Assembler verarbeitet werden.

Natürlich können Sie sich PowerWindows auch auf irgendwelchen dunklen Kanälen besorgen. Aber wenn Sie intensiv mit diesem Programm arbeiten, sollten Sie auch die Vorteile nutzen, die Ihnen durch den rechtmäßigen Erwerb entstehen: Dazu gehört das Handbuch mit der Übersicht über alle PowerWindows-Funktionen, der Update-Service sowie der Support (beispielsweise über die Inovatronics-Mailbox).

Die dunklen Kanäle

Daneben hat der Autor dieses genialen Programms Besseres verdient als eine weite Verbreitung auf dem schwarzen Markt. Oder was glauben Sie, würden Sie im Supermarkt zu hören bekommen, wenn Sie sich zum Nulltarif bedienen?

PowerWindows erhalten Sie unter anderem bei Philgerma (Barerstr. 32, 8000 München, Tel. 089/281228) für zirka 150 Mark.

Bevor Sie mit dem Workshop beginnen, sollten Sie eine formatierte Diskette bereitlegen, auf der Sie Ihre Programme speichern werden. Kopieren

Workshop – Was ist das?

In diesem AMIGA-Sonderheft finden Sie erstmals die Rubrik »Workshop«. Diese werden Sie auch in kommenden Heften immer wieder entdecken.

Wir werden unter dieser Rubrik Artikel veröffentlichen, die ähnlich wie ein Kurs intensiv bestimmte Themen behandeln. Die Workshops sollen allen Amiga-Programmierern und -Anwendern die Arbeit mit einem Programm oder einem Paket von mehreren Programmen erleichtern.

Was halten Sie von dieser Idee? Welche Vorschläge haben Sie für kommende Workshop-Themen? Wir freuen uns auf Ihre Vorschläge und Meinungen. Schreiben Sie an:

Markt & Technik Verlag AG
Redaktion Sonderhefte
Stichwort: Workshop
Hans-Pinsel-Straße 2
8013 Haar b. München

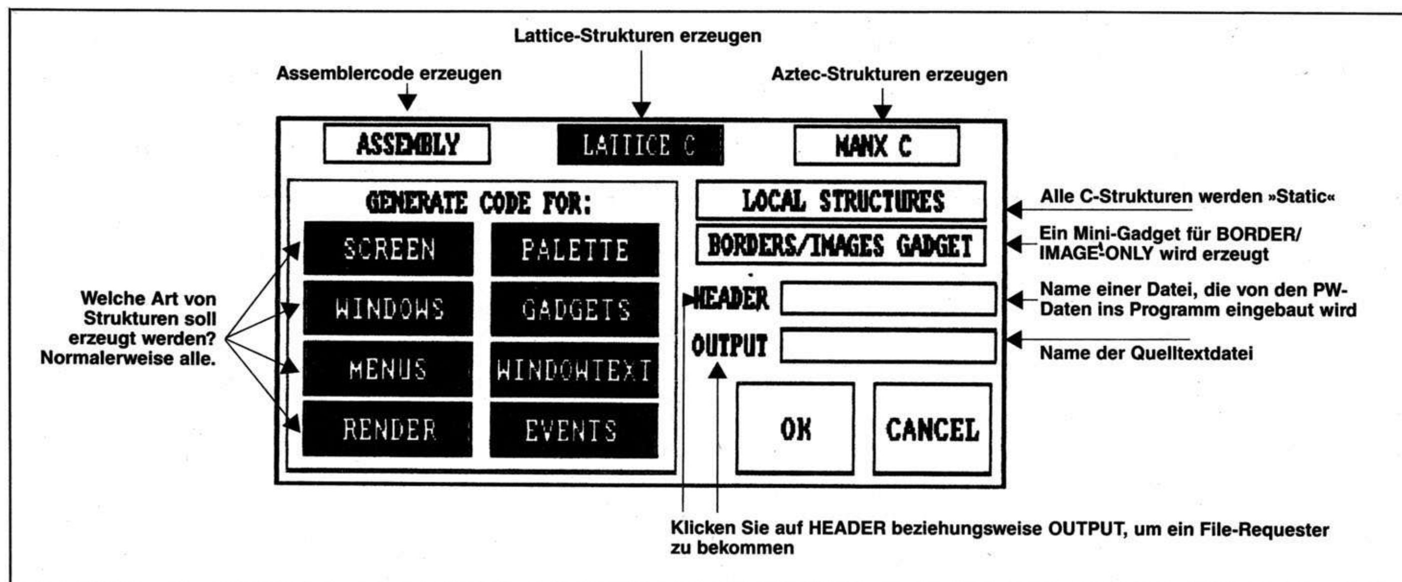


Bild 1. Sourcecode erzeugen Sie mit Hilfe dieses Steuerfelds

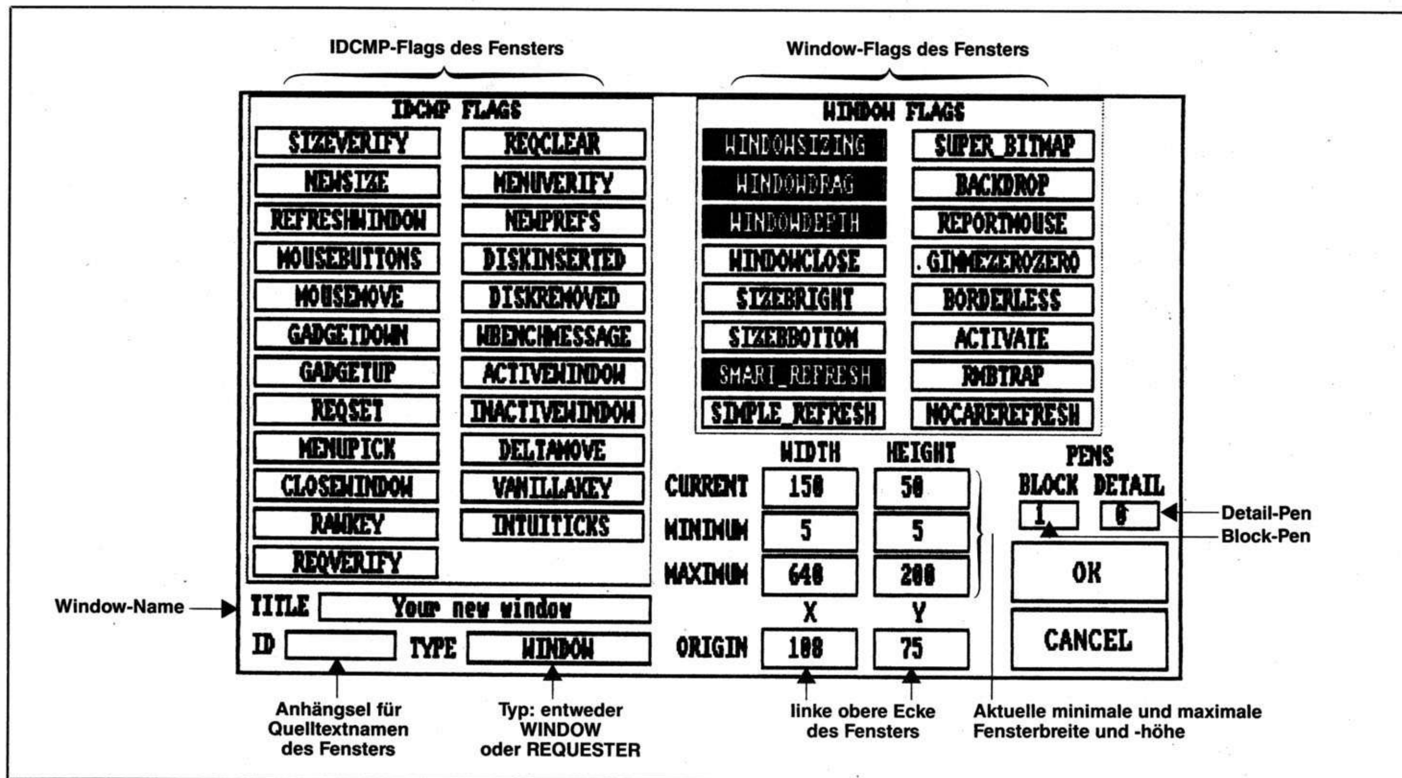


Bild 2. Sämtliche Window-Eigenschaften auf einen Blick

Sie am besten PowerWindows auf diese Diskette – es hat keinen Kopierschutz. Wenn Sie mehr als 512 KByte Speicher haben, sollten Sie PowerWindows und Ihren Compiler beziehungsweise Assembler gleichzeitig im Speicher halten, weil das Arbeiten so am angenehmsten ist. Natürlich können Sie bei diesem Workshop auch ohne Speichererweiterung mitmachen, nur müssen Sie dann PowerWindows immer wieder neu laden.

Und nun: an die Arbeit! Öffnen Sie Ihr erstes Fenster mit PowerWindows. Dazu laden Sie das Programm durch einen Klick auf sein Icon oder durch Eingabe von »PW2« im CLI. Sollten Sie bereits ein Fenster über den ganzen Bildschirm gezogen haben (beispielsweise das CLI), so werden Sie einen verblüffenden Effekt bemerken – nämlich keinen. Dies liegt daran, daß PowerWindows als sogenanntes »Backdrop«-Window definiert ist, das immer hinter allen anderen Fenstern liegt und auch nicht per Gadget hervorzuholen ist. Drücken Sie einfach die rechte Maustaste, und Sie sehen die Menüleiste. Natürlich können Sie auch alle Fenster schließen, die Ihnen im Weg sind. Wenn Ihnen diese Einstellung nicht gefällt, können Sie sie auch ändern: Gehen Sie dazu im PowerWindows-Menü »Preferences« auf »PW Backdrop« und lassen Sie die rechte Maustaste los. Darauf kommt das PowerWindows-Fenster nach vorne. Wenn Sie diese Einstellung speichern wollen, finden Sie im »Project«-Menü den Unterpunkt »Save Configuration«.

In jedem Fall sollten Sie jetzt auf Ihrem Bildschirm ein Fenster geöffnet haben, das folgenden Text in der Titelleiste anzeigt: »PowerWindows v2.0 (c)1987 by INOVATRONICS, INC.«. Natürlich kann die Versionsnummer durchaus eine andere sein. Drücken Sie jetzt die rechte Maustaste und sehen Sie sich die Menüleiste an. Neben den schon bekannten Menüpunkten »Project« und »Preferences« finden Sie vier weitere: »Screen«, »Current Window«, »Menus« und »Gadgets«. Um ein eigenes Fenster zu öffnen, wählen Sie im Menü »Current Window« den Punkt »Open a new Window«. Sofort sehen Sie ein kleines Fenster mit dem Titel »Your new Window« auf dem Schirm. Sie können es mit der Maus beliebig verschieben und in der Größe verändern. Zum Ausprobieren werden Sie es nun gleich in einem eigenen Programm einsetzen. PowerWindows ist dabei zuständig für die Datenstruktu-

```
PeO #include <intuition/intuition.h>
J7  #include <intuition/intuitionbase.h>
dH  #include <functions.h>
nk  #include "window_1.src"
Y1  struct IntuitionBase *IntuitionBase;
QQ  main ()
3W  {
bZ2   struct Window *MyWindow;  /* Zeiger auf Window-Struktur 1*
      /
QA    if (!(IntuitionBase = (struct IntuitionBase *)
Zy8         OpenLibrary ("intuition.library",0L)))
3E5      {printf ("Intuition ist nicht da! Was ist los?\n"); exi
         t (0);}
oK       /* Intuition-Library öffnen */
532   MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
      re1);
r55      /* Fenster öffnen -> MyWindow zeigt auf die Window-
HI8         Struktur. Wenn es gleich 0 ist, trat ein Fehler auf!
             */
5K2   if (MyWindow == 0) {printf ("Fenster ließ sich nicht öffne
      n!");
Y7M                        CloseLibrary(IntuitionBase);
35                         exit (0);}
OU0 Wait (1L <<MyWindow->UserPort->mp_SigBit);
13  /*  Delay (150);*/ /* Amiga-Systemroutine: drei Sekunden war
    ten */
aJ2   CloseWindow (MyWindow);   /* und das Fenster wieder zumach
      en */
KK    CloseLibrary (IntuitionBase); /* Library schließen */
Ns0 }
(C) 1988 M&T
```

Listing 1a. Dieses Steuerprogramm öffnet ein Fenster und schließt es nach etwa drei Sekunden

```
uj3    incdir   ":include/"
jX     include  intuition/intuition.i
Z1     include  intuition/intuition_lib.i
4w     include  exec/exec_lib.i
6W     include  graphics/graphics_lib.i
Vd0 NULL   EQU   0   ; wird für die PW-Definitionen benötigt
tx  * Intuition Library oeffnen:
Gd  * --------------------------
6v3    lea   intname,a1
vH     moveq   #0,d0
MC     CALLEXEC OpenLibrary
rI     tst.l   d0
CB     beq   abbruch
nY     move.l   d0,_IntuitionBase    ;Basis-Zeiger sichern
Rk0 * Window oeffnen
XU  * -------------
353    lea   NewWindowStructure1,a0          ;zeige auf Window-St
       ruktur
3h     CALLINT   OpenWindow        ;oeffne Window
sp     tst.l   d0             ;ging was schief?
7d     beq   closeint         ;wenn ja
FQ     move.l   d0,windowptr        ;Window-Zeiger sichern
zN0 * Eine Zeitlang warten
Td  * --------------------
SX3    move.l   #$ffffffff,d0         ; Zwei verschachtelte Wart
       eschleifen
eZ0 wait1:   move.l   #$f,d1
YN  wait:   dbra   d1,wait
HH8         dbra   d0,wait1
cZ0 * Fenster schliessen
WG  * ------------------
e83    move.l   windowptr,a0          ;"Window-Adresse" nach a0
qX     CALLINT   CloseWindow         ;Fenster zu
AM0 * Libraries schliessen
dn  * --------------------
Pf  closeint
cT3    move.l   _IntuitionBase,a1
rf     CALLEXEC CloseLibrary
dt0 abbruch
XD3    move.l   #0,d0                ;oder normales Ende
hi     rts
MW0 intname      INTNAME          ;Name Intuition Lib (via Makro
    )
Vi  _IntuitionBase   ds.l   1      ;Speicher fuer Zeiger
QI  windowptr   ds.l   1
vw3    include  window.src
(C) 1988 M&T
```

Listing 1b. Dieselbe Aufgabe erledigt dieses Programm in Assembler

ren, Sie müssen lediglich für die Steuerung sorgen. Lassen Sie sich also den SourceCode für Ihr Programm auf Diskette schreiben. Zu diesem Zweck wählen Sie im Project-Menü den Punkt »Generate Source Code«. Klicken Sie im darauf erscheinenden »Steuerpult« (Bild 1) mit der Maus auf den Namen Ihres C-Compilers beziehungsweise auf Assembler und dann auf das Wort »Output«. PowerWindows zeigt nun einen File-Requester, in dem Sie als Dateiname »window__1.src« eingeben. Nach einem Klick auf »OK« wird der Window-Sourcecode auf Diskette geschrieben. In Bild 1 ist kurz erläutert, was »Generate Sourcecode« an Funktionen bietet. Die genaue Wirkung entnehmen Sie bitte dem Handbuch.

Sollten Sie nur 512 KByte Speicher haben, speichern Sie das Ergebnis Ihrer Arbeit auch als PowerWindows-Datei unter dem Namen »window.pw«. Verwenden Sie dazu den Menüpunkt »Save screen and/or windows« (im »Project«-Menü). Der File-Requester, der zu diesem Zweck erscheint, ist leicht zu bedienen. Verlassen Sie dann das Programm mit »Quit«. Um bei der nächsten Sitzung die gerade gespeicherte ».pw«-Datei wieder zu laden, verwenden Sie den Menüpunkt »Load screen and/or windows«. Was steht in der Datei »window__1.src«? Betrachten Sie diese am besten eingehend mit einem Editor. Da nur ein Fenster geöffnet wurde, finden Sie auch nur eine Datenstruktur: »struct NewWindowNewWindowStructure1«. Was die einzelnen Punkte der Struktur bedeuten, hat PowerWindows für Sie als Kommentar dazugeschrieben – umständliches Nachschlagen ist überflüssig! Auf die Strukturelemente geht der C-Kurs ab Seite 24 genauer ein. Wichtig für uns ist, daß die Window-Struktur den Namen »NewWindowStructure1« trägt – dieser wird später zum Öffnen des Fensters gebraucht. Wenn Sie sich die Assembler-Datei ansehen, werden Sie feststellen, daß PowerWindows unter dem Label »NewWindowStructure1« eine Reihe von Daten abgelegt hat, die das Fenster definieren. Sie sind ebenfalls kommentiert.

Kommentierter Sourcecode

Um das Fenster in einem eigenen Programm anzuwenden, tippen Sie bitte entweder Listing 1a (für C-Programmierer) oder Listing 1b (für Assembler-Programmierer) ab. Beide Listings enthalten die Anweisung »include "window.src"«, um unsere Window-Struktur einzubauen. Beim Assembler-Programm steht diese Struktur allerdings ganz am Ende, da Sie Daten nicht mit Programm-Code mischen dürfen. Achten Sie darauf, daß »window__1.src« auch wirklich im aktuellen Directory liegt. Falls Sie im PowerWindows-File-Requester einen anderen Pfad angegeben haben, kopieren Sie das File in Ihr Arbeitsdirectory. Das Beispielprogramm öffnet das Fenster für etwa drei Sekunden und schließt es dann wieder.

Übersetzen Sie das Programm und – voilà! – das Fenster steht für einige Zeit auf dem Schirm.

Zurück zu PowerWindows. Wer mehr als 512 KByte Speicher hat, braucht nur das PowerWindows-Fenster zu aktivieren – alle anderen laden bitte das Programm wieder und holen mit dem Menüpunkt »Load screen and/or windows« im Menü »Project« das Fenster wieder auf den Bildschirm.

Bisher haben Sie nur ein ganz einfaches Fenster erzeugt. Aber lassen Sie uns jetzt gemeinsam durchleuchten, was PowerWindows mit Fenstern alles anstellen kann. Dazu wählen Sie im Menü »Current Window« den Punkt »Edit window characteristics«. Der ganze Bildschirm wird von einem »Schaltpult« für unser Fenster ausgefüllt (Bild 2). Dieses Feld dient zur komfortablen Veränderung aller Window-Eigenschaften mit der Maus. In der linken Hälfte des Feldes befinden sich die sogenannten »IDCMP-Flags« – sie entscheiden darüber, über welche Aktionen des Benutzers unser Programm informiert werden soll. Viele dieser Flags werden hier noch eine Rolle spielen. Rechts oben sehen Sie die »Window-Flags«. Sie bestimmen unter anderem das Aussehen Ihres Fensters. Als Voreinstellung sind »WINDOWSIZING«, »WINDOWDRAG«, »WINDOWDEPTH« und »SMART__REFRESH« eingeschaltet. Das

Fenster kann also in der Größe verändert, verschoben und nach hinten geklickt werden; außerdem sorgt Intuition selbständig für das Zwischenspeichern des Inhalts, falls es von einem anderen Fenster überdeckt werden sollte. Wenn Sie über die einzelnen Flags ausführliche Informationen wünschen, finden Sie diese im C-Kurs auf Seite 24 oder in einem Buch zu diesem Thema (siehe auch Seite 147).

Links unten sehen Sie den Titel des Fensters, den Sie natürlich ändern können, nachdem Sie ihn angeklickt haben. »ID« und »TYPE« sind im Moment nicht wichtig. Unten in der Mitte befinden sich die Größenwerte für das Fenster, die in »WIDTH« und »HEIGHT« (Breite und Höhe) aufgeteilt sind. Unter »CURRENT« steht der momentane Wert, »MINIMUM« enthält die kleinstmögliche, »MAXIMUM« die größtmögliche Fenstergröße. Bei »ORIGIN« sind die Koordinaten der linken oberen Ecke des Fensters zu finden. Alle Zahlen werden durch Anklicken und Eintippen des neuen Wertes geändert – oder natürlich durch Verschieben des Fensters.

Als nächstes soll ein Fenster auf den Bildschirm gebracht werden, das sich durch einen Klick auf das Close-Gadget (in der linken oberen Ecke) von selbst wieder schließt. Außerdem soll es den Titel »Amiga Sonderheft« bekommen. Tragen Sie also den Titel in das »TITLE«-Feld ein und klicken Sie rechts »WINDOWCLOSE« und links »CLOSEWINDOW«. »WINDOWCLOSE« sagt dem Amiga, daß das Fenster ein Schließ-Gadget haben soll; »CLOSEWINDOW« dagegen teilt unserem Programm mit, wenn dieses Gadget angeklickt wurde. Wenn Sie nun mit »OK« das Window-Steuerfeld verlassen, werden Sie feststellen, daß das Fenster ein Schließ-Gadget bekommen hat. Und auch der neue Titel steht in der Titelzeile des Fensters. Das Schließ-Gadget hat während der Arbeit mit PowerWindows übrigens keine Funktion.

Speichern Sie bitte wieder mit »Generate Sourcecode« den C- beziehungsweise Assembler-Sourcecode ab. Das Programm muß nun ein wenig erweitert werden, damit es auf die Meldung von Intuition, daß das Schließ-Gadget betätigt wurde, auch reagieren kann. Intuition versendet alle seine Meldungen als »Nachrichten« an das Programm (genauer gesagt an das Fenster), für das sie bestimmt sind. Das Programm braucht also ein »Postfach«, in dem diese Nachrichten abgelegt werden. Dieses Postfach hat den schönen Namen »Intuition Direct Communications Message Port«, abgekürzt »IDCMP«. Die vorher angesprochenen IDCMP-Flags sind eine Art Filter, der nur noch die vorher festgelegte Art von Nachrichten zum Postfach durchläßt. Im Moment ist nur das IDCMP-Flag »CLOSEWINDOW« gesetzt – das Programm soll nur dann informiert werden, wenn das Schließ-Gadget angeklickt wurde. Alle anderen Nachrichten behält Intuition für sich, da sie für das Programm uninteressant sind.

Wie aber werden die Nachrichten von Intuition nun empfangen? Dazu gehen Sie in drei Schritten vor:
- einen Speicherplatz für die Nachricht festlegen;
- auf eine Nachricht warten;
- auf die eingetroffene Nachricht reagieren.

Im C-Programm sieht das dann so aus, daß zuerst einmal mit

```
struct Intui
Message *msg
```

ein Speicherplatz für die Nachricht festgelegt wird. Dann wartet man mit

```
Wait (1L << MyWindow-
>UserPort->mp_SigBit);
```

auf die Nachricht. Die genaue Bedeutung dieser kompliziert wirkenden Anweisung ist unwichtig. Es genügt zu wissen, daß auf eine Nachricht gewartet wird, die beim Fenster »MyWindow« ankommen soll. Während des Wartens hat der Amiga Zeit, sich anderen Dingen zuzuwenden – er »weckt« das Programm erst, wenn eine Nachricht angekommen ist. Nun wird das Fenster geschlossen und das Programm beendet.

In Assembler funktioniert die Sache ähnlich: Mit

```
move.l MyWindow,a0
move.l wd_UserPort(a0),
       a0
move.l a0,a5
move.b MP_SIGBIT(a0),d1
moveq #0,d0
bset d1,d0
CALLEXEC Wait
```

wartet das Programm ebenfalls, bis eine Nachricht eingetroffen ist. Um dies auszuprobieren, ersetzen Sie die »Delay«-Anweisung von Listing 1 durch

```
Wait (1L << MyWindow-
>UserPort->mp_
SigBit));
```

Schon läßt sich das Fenster auf Wunsch wieder schließen.

Der Fensterklau geht um

PowerWindows bietet noch einige weitere Funktionen. Sehr interessant ist »Grab a window«. Damit können Sie nach einem Fenster eines anderen Programms »greifen« und es mitsamt allen Gadgets und Menüs in PowerWindows übernehmen. Nun wird es wie ein von PowerWindows erzeugtes Fenster behandelt, man kann es also verändern und dann den Quellcode auf Diskette schreiben lassen!

Aber auch zum Erkennen von Window-Programmiertricks ist »Grab a Window« sehr praktisch. Oder wußten Sie schon, daß alle Operationen der Workbench in einem sogenannten »Backdrop-Borderless«-Window ablaufen (und keineswegs direkt auf dem Screen-Rastport, wie immer wieder zu lesen ist)? Wenn Sie »Grab a window« anwählen, wird von PowerWindows ein neues Menü mit den Menünamen »Screens« und »Other« aktiviert. Bei »Screens« sehen Sie die Namen aller aktiven Screens als Menüpunkte; die

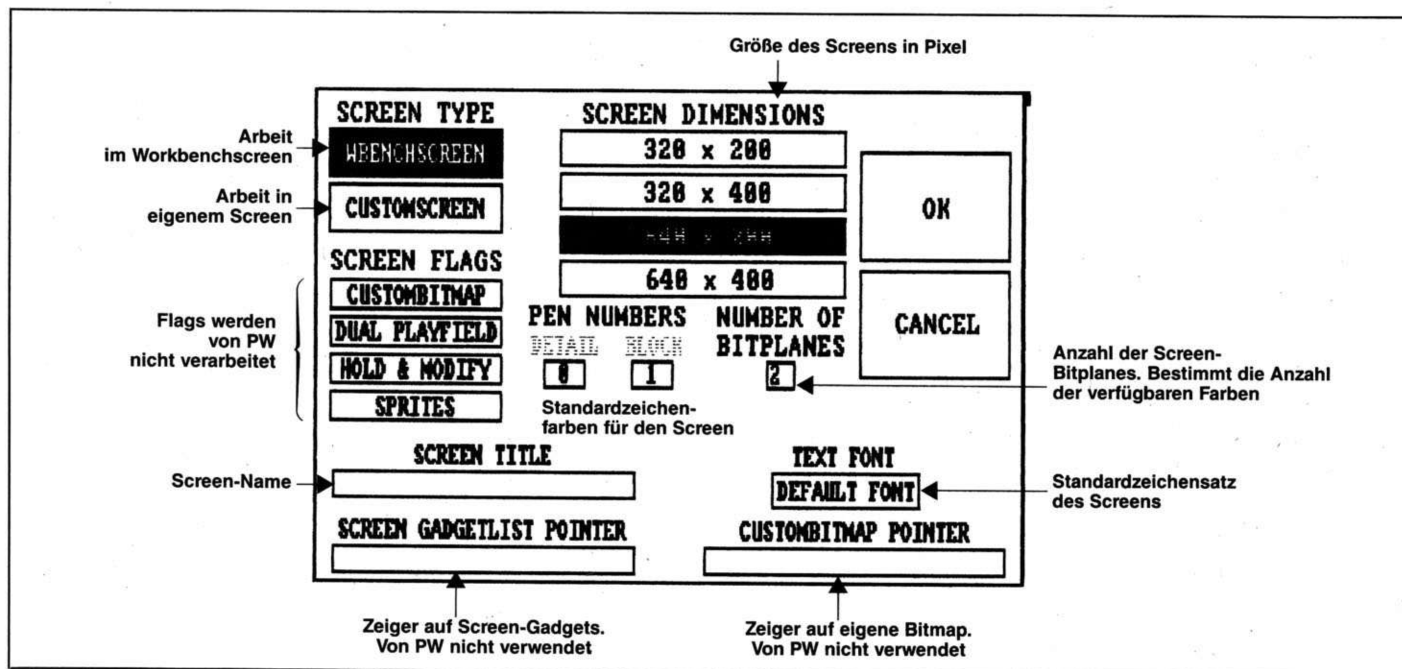


Bild 3. Customscreens können in der Version 2.0 nur bis zur (amerikanischen) NTSC-Auflösung erzeugt werden

Titelzeilen aller Fenster auf jedem Screen erscheinen als Untermenüpunkte des jeweiligen Screens. Fenster ohne Titel tauchen im Untermenü als Leerzeile auf. Um ein bestimmtes Fenster zu »greifen«, wählen Sie es einfach aus dem Menü aus. Im Menü »Other« kommen Sie mit dem einzigen Menüpunkt »Return to main menu« wieder ins Hauptmenü zurück.

Mit »Write window text« schreiben Sie beliebige Texte in Ihr Fenster. In der Abfragebox tippt man den gewünschten Text einfach ein. Klicken Sie mit der Maus auf Ihr Fenster, um den Text darin zu positionieren. Bewegen Sie ihn an die gewünschte Stelle und befestigen Sie ihn per Mausklick. Geben Sie so viele Textzeilen ein, wie Sie wollen. Beim Positionieren eines Textes schaltet PowerWindows übrigens alle anderen Texte aus – sie erscheinen wieder, sobald Sie den Text durch einen Mausklick »befestigt« haben. Mit »Edit Window Text« verändern Sie Ihre Texte: Alle bisher eingegebenen Textzeilen stehen in einem Untermenü, aus dem man auswählt. »Move window text« dient zum Verschieben von Textzeilen, und mit »Delete window text« verschwindet die angewählte Zeile wieder. Um die Texte auch mit dem Programm anzuzeigen, bauen Sie eine kleine Änderung ein: Setzen Sie bitte die Zeile

```
PrintIText(MyWindow->
RPort,&IText1,0,0)
```

in das C-Programm nach Zeile 15 ein. Nun werden alle vorher definierten Texte angezeigt.

Eigene Screens in allen Größen

Die letzte Funktion im »Current Window«-Menü ist »Kill window«, mit der Sie nach einer Sicherheitsabfrage das Window wieder löschen können.

Übrigens: Sie können natürlich auch mit mehreren Fenstern gleichzeitig in PowerWindows arbeiten, wenn Sie den Menüpunkt »Open new Window« einfach mehrmals anwählen. Beachten Sie dabei, daß für PowerWindows das aktuelle Fenster immer dasjenige ist, das Sie als letztes definiert haben. Sie sollten also immer die Arbeit mit einem Fenster abgeschlossen haben, bevor Sie das nächste definieren. »Edit window characteristics«, die Textfunktionen und »Kill window« beziehen sich also immer auf das Fenster, das als letztes definiert wurde.

Das Preferences-Menü

OK prompts läßt Sie die »OK«-Requester bei den Gadget-und Textfunktionen abschalten. Mit <HELP> läßt sich ein solcher Requester auch bei abgeschalteten »OK prompts« auf den Bildschirm bringen.

PowerWindows backdrop stellt ein, ob es sich beim PowerWindows-Fenster um ein Backdrop-Fenster handelt. Ein Backdrop-Fenster liegt hinter allen anderen Fenstern, was für manche Arbeiten ganz praktisch sein kann.

Auto redraw hält PowerWindows davon ab, ein Window nach jeder Veränderung neu zu zeichnen.

Auto select. Ist dieser Menüpunkt aktiv, wird bei jedem Mausklick außerhalb der Titelleiste eines eigenen Fensters das PowerWindows-Fenster aktiviert.

Mouse coordinates wählt, ob beispielsweise bei Gadget-Operationen die Mauszeiger-Position angezeigt werden soll.

Gadget collision checking entscheidet, ob PowerWindows bei den Gadget-Befehlen auf sich überlappende Gadgets prüft oder nicht.

Image color remapping tritt in Aktion, falls ein geladenes Image nicht dieselbe Farbpalette wie der aktive Screen hat. PowerWindows versucht dann, die Farben so gut wie möglich anzupassen.

Wenn **Image compression** aktiv ist, werden alle Bitplanes von Grafiken, die nur Nullbytes enthalten, automatisch entfernt.

Trim extra bitplanes reduziert jede Grafik auf die Zahl der Bitplanes, die der Screen hat, in dem sich die Grafik befindet.

Source code comments schaltet die Kommentare im Quelltext ein und aus.

Source code spacing hat ein Untermenü, in dem die Einrückung der Quelltextzeilen von PowerWindows eingestellt wird. Von einem bis acht Leerzeichen und einem Tabulator ist alles möglich.

PowerWindows unterstützt natürlich auch die Verwendung eigener Screens. Dazu wählen Sie im »Screen«-Menü »Define screen type«. Das Screen-Steuerfeld (Bild 3) erscheint auf dem Bildschirm. Unter »SCREEN TYPE« klicken Sie auf »CUSTOMSCREEN«, um einen eigenen Screen zu erzeugen. Bei »SCREEN DIMENSIONS« geben Sie die Screen-Größe an. Die Version 2.0 von PowerWindows hat hier allerdings ein Problem: Da die PAL-Auflösung des Amiga hier noch nicht unterstützt wird, kann man mit dieser Version auch keine PAL-Screens erzeugen. Bei der neuen Version 2.5 ist dieses Problem behoben – hier können Sie Ihre Screen-Größe vollkommen frei wählen. Im Screen-Steuerfeld gibt es noch einige andere Einstellungen: die Zahl der Bitplanes; die Standard-Zeichenfarben für den Screen; der Standardzeichensatz; der Screen-Titel und einige Screen-Flags, die PowerWindows aber nicht anwendet, sondern lediglich in den Quelltext einbaut.

Alle Fenster, die Sie nach der Screen-Definition öffnen, werden auf dem neuen Screen geöffnet. Alle »alten« Fenster werden automatisch dorthin kopiert. Wenn Sie während der Anzeige des Screens die rechte Maustaste drücken, erhalten Sie das PowerWindows-Menü. Sie können auch mit <F1> und <F2> zwischen der Workbench und Ihrem Screen hin- und herschalten. Um einen selbst erzeugten Screen wieder auszuschalten, klicken Sie im Screen-Steuerfeld wieder auf »WBENCHSCREEN«. Alle Fenster des Custom-Screens werden dann automatisch in den Workbenchscreen kopiert.

Mit »Screen Palette« wird die Farbpalette für den Screen verändert. »READ PALETTE FROM IFF« läßt Sie eine Palette von einem IFF-Bild einlesen. Für einen File-Requester zu diesem Zweck klicken Sie auf den Text »READ PALETTE FROM IFF«.

Zur Übung erzeugen Sie am besten einen Screen mit der Größe 320 x 200. Öffnen Sie in ihm ein Fenster, bei dem CLOSEWINDOW und WINDOWCLOSE gesetzt sind, und speichern Sie den Quellcode als »Screen_1.src«.

Wie bringen Sie so einen Screen auf den Bildschirm? Das Vorgehen unterscheidet sich nur wenig von Listing 1. Sie verwenden den Funktionsaufruf »OpenScreen (screen)«, der einen neuen Screen öffnet und als Wert die Adresse seiner Screen-Struktur zurückgibt. »screen« ist ein Zeiger auf die NewScreen-Struktur, bei PowerWindows hat sie den Namen NewScreenStructure1. Damit haben Sie alle Informationen, die Sie benötigen. Versuchen Sie, wenn alles geklappt hat, noch etwas mit dem eben Gelernten zu spielen. So könnten Sie beispielsweise die Farben oder die Flags verändern. Beachten Sie dabei immer, wie sich der von PowerWindows produzierte Sourcecode verändert.

Im Programm wird der Screen geöffnet. Damit das Fenster auch »weiß«, in welchem Screen es erscheinen soll, wird nun der Screen-Zeiger in der »NewWindow«-Struktur auf den gerade geöffneten Screen gestellt. Das sollten Sie nicht vergessen, sonst kann das Fenster nicht geöffnet werden! Am Ende des Programms werden Fenster und Screen wieder geschlossen.

Menüs? Ganz einfach!

Nachdem Sie nun bis hierher gelesen haben, werden Sie vielleicht denken: »Also Fenster und Screens kann ich auch ohne PowerWindows auf den Bildschirm bringen!« Deshalb kommt nun etwas sehr viel Komplizierteres. Haben Sie schon einmal versucht, ein Programm mit einem Menü zu versehen? Dann wissen Sie, daß das eine sehr aufwendige Sache ist, bei der man vor lauter »MenuItem«- und »IntuiText«-Strukturen leicht den Überblick verliert. Bei PowerWindows haben Sie damit nichts zu tun! Als Beispiel soll ein Menü für einen einfachen Texteditor dienen. Es besteht aus zwei Menü-»Überschriften«: »Projekt« und »Text«. Im Menü »Projekt« sollen die Menüpunkte »Text laden«, »Text speichern«, »Info« und »Ende« stehen. Das »Text«-Menü soll die Punkte »Ausschneiden«, »Kopieren«, »Einsetzen«, »Bewegen« und »Löschen« bekommen.

Öffnen Sie also mit PowerWindows ein Fenster und setzen Sie die Flags »CLOSEWINDOW« und »WINDOWCLOSE« sowie »MENUPICK«. Nun wählen Sie aus dem PowerWindows-Menü »Menu« den ersten Punkt, »Append a Menu« (ein Menü anhängen). Es erscheint eine Abfragebox, in der Sie das Wort »Projekt« eingeben und auf »OK« klicken. Um das Menü anzusehen, klicken Sie auf die Titelleiste oder auf das Close-Gadget Ihres Fensters. Nach dem Druck auf die rechte Maustaste sehen Sie das Menü »Projekt« mit dem einzigen Menüpunkt »dummy« auf dem Bildschirm. Nun bekommt das »Projekt«-Menü seine Menüpunkte. Dazu müssen Sie erst PowerWindows aktivieren (Mit einem Klick ins PowerWindows-Fenster). Das ist einer der großen Vorteile dieses Pro-

gramms: Sie können sich Ihr Menü sofort durch einen einfachen Mausklick ansehen und es verändern, falls Sie nicht damit zufrieden sind.

Wählen Sie nun im »Menu«-Menü »Work on Menuitems for...«. Es erscheint ein Untermenü, dessen einziger Eintrag im Moment noch »Projekt« ist. Diesen Untermenüs werden Sie bei der Arbeit mit PowerWindows noch öfter begegnen: Sie machen die Auswahl des Menüs, das bearbeitet werden soll, sehr einfach. Hier wird in Zukunft als Abkürzung für »wählen Sie den Menüpunkt X mit dem Unterpunkt Y« einfach stehen »Wählen Sie den Punkt X/Y«.) Bewegen Sie also den Mauszeiger auf »Projekt« und lassen Sie ihn los.

Bei einem Druck auf die rechte Maustaste sehen Sie, daß eine neue Menüleiste eingeschaltet ist, die »MenuItems« und »Other« enthält. Auch das Umschalten zwischen den verschiedenen Menüs werden Sie noch öfter beobachten. Ganz allgemein dient »Other« immer dazu, wieder in das Hauptmenü (»Return to main menu«) oder in das vorherige Menü (»Return to previous Menu«) zurückzukehren – wir werden auf »Other« also nicht mehr ausdrücklich hinweisen. Wählen Sie aus dem ersten Menü »Append a menuitem« und tippen Sie in die große Abfragebox »Text laden« ein, ohne sich um den Rest der Box zu kümmern. Nach einem Klick auf »OK« ist »Text laden« der erste (und bisher einzige) Menüpunkt von »Projekt« geworden, wie Sie durch einen Klick auf die Fenster-Titelleiste leicht feststellen können. Klicken Sie wieder in das PowerWindows-Fenster und hängen Sie mit »Append MenuItem« die restlichen Menüpunkte (»Text speichern«, »Info« und »Ende«) an das Projekt-Menü an.

Über das »Other«-Menü gelangen Sie wieder in das PowerWindows-Hauptmenü. Nun erzeugen Sie mit »Append Menu« das »Text«-Menü und geben wie bei »Projekt« besprochen die Menüpunkte »Ausschneiden«, »Kopieren«, »Einsetzen«, »Bewegen« und »Löschen« ein. Wenn Sie sich Mausbewegungen sparen wollen, sollten Sie sich die Shortcuts neben den einzelnen Menüpunkten ansehen.

Sollten Sie sich bei einem Menüpunkt einmal vertippt haben, stehen die PowerWindows-Menüpunkte »Edit a menu« und »Edit a menuitem« für Menüs und Menüpunkte zur Verfügung. Hier können Sie das fehlerhafte Menü oder den fehlerhaften Menüpunkt – wie üblich über ein Untermenü – auswählen und korrigieren. Wollen Sie beispielsweise an einem Menüpunkt von »Projekt« arbeiten, müssen Sie natürlich erst mit »Work on menuitems.../Projekt« in das Menuitem-Menü umschalten.

Wie aber baut man nun das Menü in ein eigenes Programm ein? Ein Menü soll nicht einfach nur angezeigt werden, sondern das Programm muß auch davon erfahren, wenn ein Menüpunkt ausgewählt wurde. Damit dies korrekt geschieht, müssen Sie unbedingt das Flag »MENUPICK« im Fenster-Steuerfeld setzen. Es sagt Intuition, daß Ihr Programm »wissen will«, wenn ein Menü gewählt wurde. Erzeugen Sie dann bitte die Datenstrukturen unseres Menüs in Ihrer Lieblings-Programmiersprache. Bevor Sie PowerWindows beenden, speichern Sie bitte mit »Save Windows and/or Screens« (ist im »Project«-Menü von PowerWindows zu finden) die Ergebnisse Ihrer Arbeit, damit wir später mit demselben Menü weiterarbeiten können.

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <functions.h>
#include "Menu.src"
struct IntuitionBase *IntuitionBase;
main ()
{
  struct Window *MyWindow;
  struct IntuiMessage *Message;
  int class, code;
  if (!(IntuitionBase = (struct IntuitionBase *)
          OpenLibrary ("intuition.library",0L)))
    {printf ("Intuition ist nicht da! Was ist los?\n"); exi
    t (0);}
  MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
  re1);
    if (MyWindow == 0) {printf ("Fenster ließ sich nicht öff
    nen!");
                          CloseLibrary(IntuitionBase);
                          exit (0);}
  SetMenuStrip (MyWindow,&Menu1); /* Menü einschalten */
  do {  /* Diese Schleife wird durchlaufen, bis eine CLOSEWI
  NDOW-
           Nachricht kommt.  */
   Wait (1L << MyWindow->UserPort->mp_SigBit); /* Auf (i
   rgend)eine
                              Nachricht warten */
    while (Message = GetMsg (MyWindow->UserPort))
     {
      class = Message -> Class;  /* Class- und Code-Wert  *
      /
      code = Message -> Code;    /* speichern             *
      /
      ReplyMsg (Message);          /* Nachricht quittieren  */
       if (class == CLOSEWINDOW) break; /* Fenster zu! */
       if (class == MENUPICK)       /* Menü angewählt! */
         printf ("Ausgewählt wurde Menü %d mit dem Menüpunkt
          %d.\n",
                  MENUNUM (code), ITEMNUM (code));
                              /* Angewählte Menüpunkte ausgeben
                               */
     } /* Ende der while-Schleife */
   } while (class != CLOSEWINDOW);
  CloseWindow (MyWindow);
  CloseLibrary (IntuitionBase);
}
(C) 1988 M&T
```

Listing 2. Hier sehen Sie eine einfache Menüabfrage

Der Einbau ins Programm

Um ein Menü anzuzeigen, genügt der Aufruf einer Intuition-Funktion: »SetMenuStrip (window,Menu)« zeigt im Fenster »window« das Menü »Menu« an. Mit »ClearMenuStrip (window)« kann man das Menü des Fensters Window wieder entfernen.

Wie bereits bei den Fenstern besprochen, sendet Intuition alle Informationen, die es einem Programm zukommen lassen möchte, an den »Intuition Direct Communications Message Port« des Fensters. Auf diese »Nachrichten« wird, wie Sie bereits wissen, mit »Wait« gewartet. Bisher konnte das Programm nur eine Nachricht erhalten: »CLOSEWINDOW«, also die Information, daß das Schließ-Gadget des Fensters angeklickt wurde. Wird ein Menüpunkt ausgewählt, sendet uns Intuition ebenfalls eine Nachricht – das Programm muß also zwischen »Menüpunkt gewählt« und »Schließ-Gadget geklickt« unterscheiden können.

Eine Intuition-Nachricht enthält einige Daten, von denen nur zwei momentan wichtig sind: »Class« und »Code«. Class enthält die Art der Nachricht, beispielsweise »CLOSEWINDOW« (»Fenster zu«) oder »MENUPICK« (Menüauswahl hat stattgefunden). Code enthält bei einer Menüauswahl die (in 16 Bit verschlüsselte) Nummer des ausgewählten Menüs, Menüpunkts und Unterpunkts. Um auf eine Intuition-Nachricht zu reagieren, muß das Programm wie bisher mit »Wait« auf die Nachricht warten. Mit der Funktion »GetMsg« muß die Nachricht nun »abgeholt« werden. Class und Code werden in eigene Variablen kopiert – dann wird die Nachricht mit »Reply Msg« quittiert. Dies ist unbedingt erforderlich und soll so schnell wie möglich nach GetMsg erfolgen! Da Class und Code aber nach ReplyMsg nicht mehr lesbar sind, muß man die beiden in eigene Variablen umkopieren und dann die Nachricht quittieren.

Um eine Menüauswahl zu erkennen, muß das Programm den (zwischengespeicherten) Class-Wert mit MENUPICK vergleichen. Ist der Vergleich erfolgreich, wurde ein Menüpunkt angewählt. Welcher, steht im Code-Feld der Nachricht – allerdings bisher noch verschlüsselt. Die Entschlüsselung geht folgendermaßen:

```
Nummer_des_gewählten_
Menüs = MENUNUM (Code);
Nummer_des_Menüpunktes
= ITEMNUM (Code);
Nummer_des_Unterpunktes
= SUBNUM (Code);
```

Menüs, Menüpunkte und Unterpunkte werden mit 0 beginnend durchnumeriert.

Wie man das alles in ein Programm einbaut, sehen Sie in Listing 2. Es wartet auf eine Menüauswahl und druckt die Nummer des Menüs und des Menüpunkts in das CLI-Fenster. Dabei wurde noch eine »while«-Schleife eingebaut, um auch wirklich alle Nachrichten von Intuition zu lesen. In Ihren Programmen sollte das genauso sein. Im Moment wird SUBNUM übrigens noch nicht benutzt, da ja noch kein Untermenü definiert wurde.

PW formatiert Menü automatisch. Die Zahlenwerte werden ignoriert
Farbe der Menüpunkte
AUTO FORMATTING
ITEM RENDER
LEFT TOP
8 8
HITBOX
LEFT TOP
8 8
WIDTH HEIGHT
8 8
USER FORMATTING
OK
MENU FLAGS
HIGHCOMP
HIGHBOX
HIGHNONE
CHECKIT
MENUTOGGLE
ITEMENABLED
CHECKED
ROOM FOR CHECK
ROOM FOR ALTKEY
CANCEL
FRONT PEN BACK PEN
3 1
INTUITEXT CONTROL
DRAWMODE
COMPLEMENT
TEXT FONT
DEFAULT FONT
ITEM TEXT
FUNCTION
COMMAND SEQUENCE KEY
linke obere Ecke des Menütextes oder der Menügrafik
linke obere Ecke der »Hitbox«
Größe der »Hitbox«
Muß aktiviert sein, wenn eigene Werte eingegeben werden sollen
Farbe der Menüs
Zeichenmodus
Zeichensatz
Name des Menüpunktes
Taste für Shortcut

Bild 4. So einfach bauen Sie Ihre Menü-Items zusammen

Vielleicht haben Sie schon gemerkt, daß Intuition dem Programm auch dann eine Menü-Nachricht schickt, wenn die rechte Maustaste gedrückt und wieder losgelassen wurde, ohne daß man ein Menü gewählt hatte. Die Anzeige sieht dann so aus:

```
Ausgewählt wurde Menü
63 mit dem Menüpunkt 31.
```

Dieser Fall muß natürlich abgefangen werden. Dazu gibt es die Intuition-Konstante »MENUNULL«. Wenn Code gleich MENUNULL ist, wurde kein Menüpunkt ausgewählt. In Listing 2 müssen Sie also in Zeile 30 schreiben:

```
if (code != MENUNULL)
printf...
```

Experimentieren Sie ruhig ein wenig mit dem Programm. Natürlich ist es nicht der Sinn eines Menüs, eine Anzeige wie »Der Menüpunkt X wurde ausgewählt« auf den Bildschirm zu bringen, sondern man will einen bestimmten Programmteil starten. Am einfachsten geht das mit einer »case«-Abfrage.

Menüs entschlüsselt

Aber zurück zu PowerWindows. Sie haben nämlich lange nicht alle Möglichkeiten kennengelernt, die das Programm für Menüs bietet. Oft ist es praktisch, wenn man statt dem Aufruf eines Menüpunktes mit der rechten Maustaste einfach die rechte Amiga-Taste zusammen mit einem bestimmten Buchstaben drücken kann. Einige dieser »Tasten-Abkürzungen« sollen auch in unser Menü eingebaut werden, und zwar für »Ausschneiden« ein X, für »Kopieren« ein C und für »Einsetzen« ein P. Dazu laden Sie bitte wieder PowerWindows und die abgespeicherte PowerWindows-Datei mit dem Menü. Wählen Sie nun »Work on menuitems for... / Text«. Nun bringen Sie mit »Edit Menuitem / Ausschneiden« das Menu-Item-Steuerfeld (Bild 4) auf den Bildschirm. Rechts unten geben Sie den gewünschten »Command Key«, also hier ein X, ein. Nun klicken Sie noch auf »Room for Altkey«, damit für die im Menü erscheinende Tastenkombination auch Platz vorhanden ist. Schon hat »Ausschneiden« seine eigene Tastenkombination, wie Sie leicht selbst nachprüfen können. Wenn Sie nun das neue Menü in das Programm einbauen, werden Sie sehen, daß die Betätigung einer Tastenkombination für das Programm wie die Auswahl des entsprechenden Menüpunktes aussieht. Am Programm muß also nichts geändert werden.

Das Menu-Item-Steuerfeld bietet noch eine Fülle von Funktionen. Einige von ihnen werden wir in diesem Kurs noch anwenden; im Bild 4 sind sie kurz erklärt.

Nehmen wir an, der Texteditor würde noch etwas erweitert. Er kann jetzt auch mit verschiedenen Druckerzeichensätzen arbeiten – und die sollen per Menü gesteuert werden. Die Zeichensätze heißen »Times Roman«, »Courier«, »Helvetica« und »Pica«. Damit der Benutzer des Programms immer weiß, welcher Zeichensatz aktiv ist, soll neben dem gewählten Zeichensatz ein Häkchen stehen. Und natürlich kann man nicht mit zwei Zeichensätzen gleichzeitig arbeiten! Kompliziert? Nicht mit PowerWindows. Wie Sie sicher wissen, kann Intuition automatisch ein kleines Häkchen vor einen Menüpunkt setzen, wenn dieser gerade aktiv ist. Man nennt solche Menüpunkte, die ein Häkchen haben können, »Attribute-Items«, weil sie eine Eigenschaft anzeigen. Alle Items ohne Häkchen heißen »Aktions-Items«, weil sie eine Aktion wie das Laden eines Textes auslösen.

Damit immer nur ein Zeichensatz ein Häkchen bekommt, benutzt Intuition das »Mutual Exclusion«-System. Auf deutsch heißt das »gegenseitiger Ausschluß«. Es bedeutet, daß Sie für jeden Menüpunkt sagen können, bei welchen anderen das Häkchen gelöscht werden soll, wenn man diesen Menüpunkt wählt. Damit ist das Problem schnell gelöst:

Erzeugen Sie zunächst einmal das neue Menü »Zeichensatz«. Danach schalten Sie wie gehabt mit »Work on Subitems for... / Zeichensatz« auf die Bearbeitung dieses Menüs um. Nun wird mit »Append menuitem« der erste Menüpunkt angehängt: »Times Roman«. Damit Intuition weiß, daß dieser Punkt beim späteren Aufruf ein Häkchen bekommen soll, aktivieren Sie bitte »CHECKIT« (»gib ihm ein Häkchen«) und »ROOM FOR CHECK«. Letzteres Flag sagt PowerWindows, daß es Platz für ein Häkchen reservieren soll. Zusätzlich aktivieren Sie bitte noch »CHECKED« (»es hat ein Häkchen«), damit der erste Menüpunkt bereits beim Start des Programms mit einem Haken versehen ist.

Bei den restlichen Menüpunkten (»Courier«, »Helvetica« und »Pica«) verfahren Sie genauso. Schalten Sie aber ab Courier das CHECKED-Flag aus, sonst sind später alle Zeichensätze aktiv!

Völlig ausgeschlossen

Wenn Sie das neue Menü ausprobieren, werden Sie feststellen, daß wir noch keinen »gegenseitigen Ausschluß« eingebaut haben – man kann nämlich im Moment alle Zeichensätze gleichzeitig mit einem Häkchen versehen. Aktivieren Sie also mit »Edit menuitem... / Times Roman« noch einmal das Steuerfeld für den ersten Menüpunkt. Drücken Sie die rechte Maustaste – es erscheint ein Menü mit dem Namen »Mutual Exclusion«. Der einzige Menüpunkt »Pick items to exclude« enthält als Unterpunkte die vier Zeichensätze. Geben Sie jedem außer »Times Roman« ein Häkchen. Das bedeutet, daß bei der Auswahl von »Times Roman« alle anderen Häkchen automatisch ausgeschaltet werden.

Mit den anderen Menüpunkten verfährt man genauso – es werden bei jedem Punkt alle anderen ausgeschlossen. (Übrigens: Ein Punkt kann sich auch selber ausschließen, aber dann kann man ihn nie anwählen...) Nun funktioniert das Ganze, wie es soll – nur ein Zeichensatz kann eingeschaltet sein. Wenn Sie wollen, probie-

ren Sie das einmal mit dem Menü-Auswertprogramm aus; es sind keine Programmänderungen erforderlich.

Wenn Sie in Zukunft mit »Mutual Exclusion« arbeiten, ein Tip: Geben Sie immer zuerst alle Menüpunkte ein und kümmern Sie sich danach um den gegenseitigen Ausschluß. Anders geht es schon deshalb nicht, weil PowerWindows ja bei der Eingabe des ersten Menüpunktes den dritten noch gar nicht kennt.

Viele Textprogramme können nicht nur ASCII-Dateien, sondern auch andere Textformate lesen. Doch wie sagt man dem Programm, welches Format es lesen soll? Am einfachsten geht das mit einem Untermenü für »Text laden«. Es soll die Menüpunkte »ASCII«, »Beckertext« und »UBM-Text« enthalten. Mit PowerWindows ist das kein Problem. Schalten Sie zunächst mit »Work on menuitems for... / Projekt« auf die Bearbeitung des Projekt-Menüs. Dann wählen Sie »Work on Subitems for... / Text laden«. PowerWindows zeigt ein Menü an, das dem vorhergehenden sehr ähnlich sieht. Es hat nur einen neuen Menüpunkt: Mit »Number of columns« können Sie Ihre Untermenüpunkte in bis zu neun Spalten anordnen. Geben Sie nun »ASCII«, »Beckertext« und »UBM-Text« mit »Append Subitem« als Untermenüpunkte ein. Wenn Sie das Menü ausprobieren, sehen Sie die neuen Unterpunkte, die von PowerWindows automatisch an die richtige Stelle gesetzt wurden.

Um die Nummer des angewählten Unterpunkts abzufragen, müssen wir unser Programm etwas ändern. Die Zeile 35 wird zu:

```
printf ("Menü: %d Item:
%d SubItem: %d", MENUNUM
(code),ITEMNUM(code),
SUBNUM(code);
```

Auf diese Weise wird nun auch der gewählte Untermenüpunkt angezeigt. Falls kein Unterpunkt existiert, so hat SUBNUM(code) den Wert der Konstanten »NOSUB«. Untermenüs können genau wie normale Menüs mit Häkchen und Tastatur-Kommandos versehen werden. Probieren Sie es aus, es gibt viele Möglichkeiten!

Ein weiteres wichtiges Menü-Flag ist »MENUTOGGLE«. »To toggle« bedeutet »hin- und herschalten«. Dieses Flag ist nur bei Attribut-Items (denen mit dem Häkchen) sinnvoll. Ist es bei einem Menüpunkt gesetzt, so wird das Häkchen bei jedem Anwählen des Menüs umgeschaltet. Normalerweise ist das nicht der Fall – haben Sie schon einmal versucht, einen Zeichensatz durch Anwählen seines Namens wieder auszuschalten? Es geht nicht.

Ein gutes Beispiel für den Einsatz von MENUTOGGLE ist ein Textprogramm mit verschiedenen Schriftstilen wie Kursiv, Fett, Unterstrichen etc. Sie sollen über ein Menü ausgewählt werden und sich dort jederzeit beliebig ein- und ausschalten lassen. Erzeugen Sie also bitte ein neues Menü namens »Schrift« mit den Menüpunkten »Kursiv«, »Unterstrichen«, »Fett«, »Hochgesetzt« und »Tiefgesetzt«. Dabei setzen Sie bei »Kursiv« die Flags CHECKIT, MENUTOGGLE und ROOM FOR CHECK. Bei den anderen Menüpunkten muß nur noch der Name eingegeben werden, weil PowerWindows die Flags automatisch so einstellt wie bei der letzten Eingabe. Schon haben Sie ein Schriftenmenü, in dem jede Schrift beliebig ein- und auszuschalten ist.

Geisternde Menüs

Bei der Abfrage des neuen Menüs durch das Programm kann man vorgehen wie bisher. Trotzdem wird das Programm geändert: Es gibt die bisher nicht berücksichtigte Möglichkeit, daß man mehrere Menüpunkte gleichzeitig auswählt, was beispielsweise bei Schriftarten recht häufig sein dürfte. Das haben Sie noch nie probiert? Na, dann halten Sie mal während der Menüauswahl wie üblich die rechte Taste fest und drücken beim gewünschten Menüpunkt auch noch die linke. Schon kann man beliebig viele Häkchen aus- und einschalten. Um diesen Fall auch im Programm zu erkennen, gibt es das neue Abfrageprogramm in Listing 3. Dies ist übrigens die von Commodore-Amiga vorgeschlagene Standard-Menüabfragetechnik. Sie sollten ab jetzt nur mit dieser Methode arbeiten.

In manchen Fällen kann es auch praktisch sein, direkt »bei Intuition anzufragen«, ob ein bestimmter Menüpunkt gerade ein Häkchen hat oder nicht. Die Funktion »IsChecked (Window, Menu,Item,Sub)« erledigt das für Sie (Listing 4). Die Parameter der Funktion sind im Listing erklärt.

Eine wichtige Funktion von Intuition ist das »Ghosting«, also das Schreiben eines Menüs, Menüpunktes oder Untermenüpunktes in »Geisterschrift«, die deutlich macht, daß dieses Menü-Element im Moment nicht gewählt werden kann. Wenn Sie von vorne herein bestimmte Menü-Elemente nicht wählbar machen wollen, verwenden Sie dazu das »MENUENABLED«-Flag (für ganze Menüs) beziehungsweise das »ITEMENABLED«-Flag (für Menüpunkte und Untermenüs). Diese Flags stehen in den jeweiligen Steuerfeldern für Menüs, Menüpunkte und Untermenüs (Bild 4 und 5). Ist MENUENABLED oder ITEMENABLED ausgeschaltet, so kann der von diesem Flag betroffene Bereich nicht mehr angewählt werden. Dabei gilt:

- Wenn Sie ein ganzes Menü abschalten, sind alle Menüpunkte und Unterpunkte dieses Menüs ebenfalls abgeschaltet.
- Wenn Sie einen Menüpunkt abschalten, werden alle eventuell vorhandenen Unterpunkte abgeschaltet.

```
 1 Pe0 #include <intuition/intuition.h>
 2 J7  #include <intuition/intuitionbase.h>
 3 dH  #include <functions.h>
 4 g4  #include "Menu_1.src"
 5 Y1  struct IntuitionBase *IntuitionBase;
 6 QQ  main ()
 7 3W  {
 8 LK2   struct Window *MyWindow;
 9 sC    struct IntuiMessage *Message;
10 NS    int class, code;
11 oy    struct MenuItem *Item;
12 TD    if (!(IntuitionBase = (struct IntuitionBase *)
13 eAA           OpenLibrary ("intuition.library",0)))
14 6H5      {printf ("Intuition ist nicht da! Was ist los?\n"); exi
            t (0);}
15 752   MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
         re1);
16 5K4     if (MyWindow == 0) {printf ("Fenster ließ sich nicht öff
           nen!");
17 Y70                         CloseLibrary(IntuitionBase);
18 35                          exit (0);}
19 812   SetMenuStrip (MyWindow,&Menu1); /* Menü einschalten */
20 yU    do {  /* Diese Schleife wird durchlaufen, bis eine CLOSEWI
         NDOW-
21 JnB            Nachricht kommt.  */
22 nx3    Wait (1L << MyWindow->UserPort->mp_SigBit); /* Auf (i
          rgend)eine
23 90C             Nachricht warten */
24 Jr4     while (Message = GetMsg (MyWindow->UserPort))
25 Lo5      {
26 kZ6       class = Message -> Class;  /* Class- und Code-Wert  *
             /
27 dr        code = Message -> Code;    /* speichern            *
             /
28 Qm        ReplyMsg (Message);        /* Nachricht quittieren  */
29 hy7        if (class == CLOSEWINDOW) break; /* Fenster zu! */
30 Kv         if (class == MENUPICK)      /* Menü angewählt! */
31 sv8         while (code != MENUNULL) {
32 UR9          Item = ItemAddress (&Menu1,code); /* Strukturadress
                e des
33 bMk                                           letzten Menüpu
                                                 nktes
34 AJ                                            holen */
35 jN9          printf ("Ausgewählt wurde Menü %d mit dem Menüpunkt
                 %d und dem Unterpunkt %d.\n",
36 OTI                  MENUNUM (code), ITEMNUM (code), SUBNUM(cod
                        e));
37 SLR                           /* Angewählte Menüpunkte ausgeben
                                  */
38 Ww9          code = Item->NextSelect;  /* Code des nächsten Men
                ü-
39 JYc                                    punktes. Falls keiner
                                          da ->
40 xG                                     code == MENUNULL */
41 TD8         } /* Schleife wird beendet, wenn kein weiterer Menüp
               unkt
42 hrD             ausgewählt wurde. */
43 Cm6       } /* Ende der while-Schleife */
44 UO3    } while (class != CLOSEWINDOW);
45 Ua2   CloseWindow (MyWindow);
46 nu    CloseLibrary (IntuitionBase);
47 1GO }
(C) 1988 M&T
```

Listing 3. Diese Technik verwenden Sie als Standard-Menüabfrage.

```
BOOL IsChecked (MenuAdr,Menu,Item,Sub)   /* By Andy Lietz 5.2.89 */
/* Diese Funktion prüft, ob ein bestimmter Menüpunkt im Moment mit
   einem Häkchen versehen ist. Wenn ja, gibt sie TRUE zurück,
   sonst FALSE.
   Parameter: MenuAdr  Adresse des Menüs, normalerweise &Menu1.
              Menu     Nummer des Menüs
              Item     Nummer des Menüpunktes
              Sub      Nummer des Unterpunktes. Wird ignoriert,
                       falls der jeweilige Menüpunkt keine
                       Unterpunkte hat.
   Ein typischer Aufruf der Funktion sieht so aus:
    if (IsChecked(&Menu1,1,1,0) printf ("Häkchen gefunden!\n")
   etc.
             */
struct Menu *MenuAdr;
int Menu;
int Item;
int Sub;
{
  struct MenuItem *TempItem;
  TempItem =
       ItemAddress (MenuAdr,SHIFTMENU(Menu)|SHIFTITEM(Item)|SHIFTSUB(Sub));
   if ((TempItem->Flags & CHECKED) == CHECKED) return (TRUE);
     else return(FALSE);
}
(C) 1988 M&T
```

Listing 4. Die Funktion »IsChecked« setzen Sie ein, wenn Sie wissen wollen, ob ein Menüpunkt einen »Haken« hat

```
int MenuVal (menu,item,sub)
/* Diese Funktion rechnet aus Menünummer, Menüpunkt und Unterpunkt den verschlüsselten Menücode aus.
   Hat der verwendete Menüpunkt keinen Unterpunkt, sollten
   Sie dafür NOSUB einsetzen.
   Mögliche Aufrufe:
   MenuCode = MenuVal (0,1,NOSUB);
   MenuCode = MenuVal (2,3,4);
   etc.                                              */
int menu;
int item;
int sub;
{
  return (SHIFTMENU (menu) | SHIFTITEM (item) | SHIFTSUB (sub));
}
(C) 1988 M&T
```

Listing 5. Die Funktion »MenuVal« errechnet aus den Menünummern den zugehörigen Code

Um auch vom Programm aus Teile des Menüs ein- und auszuschalten, gibt es die Intuition-Funktionen »OnMenu« und »OffMenu«, die beide die verschlüsselte Menünummer als Parameter brauchen. Um mit »normalen« Menü-Werten (z.B. Menü 0, Punkt 1, Unterpunkt 1) zu arbeiten, verwenden Sie zusätzlich die MenuVal-Funktion (Listing 5).

Noch etwas kann PowerWindows: langweilige Texte in Menüs durch Grafiken ersetzen. Wenn Sie einen Menüpunkt als Grafik gestalten wollen, müssen Sie die Grafik zeichnen und als IFF-Pinsel auf Diskette speichern (die Größe des Pinsels bestimmt auch die Größe des Menüpunktes). Nun wählen Sie »Work on Images for...« mit dem Menüpunkt, der durch eine Grafik ersetzt werden soll. Es erscheint ein neues Menü. Mit »Load item render image« laden Sie eine Grafik für den nicht aktivierten Zustand des Menüpunktes; »Load item select image« lädt eine Grafik für den aktivierten Zustand. Wenn Sie nur »Load item render image« verwenden, wird die Grafik – wie normaler Text – invertiert, falls man den Mauszeiger darüberbewegt. »Delete item render imagery« und »Delete item select imagery« schalten die Grafiken für den jeweiligen Zustand wieder ab. Für Untermenüpunkte funktioniert das Laden von Grafiken genauso.

Im Prinzip können Sie auch Menüs mit Grafiken ganz normal in das Programm einbauen; wenn Sie mehr als 512 KByte Speicher haben, müssen Sie beim Compilieren mit Aztec C die Linker-Option »+Cd« verwenden, sonst sehen Sie später statt der Grafik irgendwelchen Unsinn auf dem Schirm.

Das Klick-Dingsda

Mindestens genauso wichtig wie die Menüs sind die Gadgets (englisch für »Dingsda«) von Intuition. Mit ihnen kann der Benutzer durch einen einfachen Mausklick auf ein Programm Einfluß nehmen. Intuition kennt drei unterschiedliche Gadget-Typen:

- das Boolean-Gadget: einfaches ja/nein-Gadget. Es wird mit der Maus angeklickt, um irgendeine Aktion des Programms hervorzurufen;
- das String-Gadget: Es erlaubt das Eintippen von Zeichenketten, die vom Programm weiterverwendet werden;
- das Proportional-Gadget: Es enthält einen »Schieberegler«, mit dem sich wie bei einem Lautstärkeregler verschiedene Werte mit der Maus einstellen lassen.

Alle drei Typen werden von PowerWindows unterstützt.

Gadgets können grundsätzlich nur in Fenstern auftreten und werden beim Öffnen des Fensters, für das sie definiert wurden, automatisch mit angezeigt. Um mit Gadgets in PowerWindows arbeiten zu können, müssen Sie daher erst mit »Open a new window« ein Fenster öffnen, falls Sie das nicht schon getan haben. Setzen Sie bitte bei allen Gadget-Beispielen wie bisher die Window-Flags CLOSEWINDOW, WINDOWCLOSE und ACTIVATE zusätzlich zu denen, die von PowerWindows bereits eingeschaltet wurden.

Wählen Sie aus dem »Gadget«-Menü »Add a Gadget« (ein Gadget anhängen). Es erscheint eine Dialogbox, die Sie einfach mit einem Klick auf »OK« bestätigen. Nun klicken Sie mit der Maus auf die gewünschte linke obere Ecke des Gadgets, machen es per Mausbewegung so groß wie nötig (für dieses Beispiel sollte das Gadget einen großen Teil des Fensters einnehmen) und drücken Sie noch mal die linke Maustaste. Schon ist das erste Gadget fertig.

Wenn Sie in das umrahmte Feld mit der Maus hineinklicken, wird es invertiert – PowerWindows hat also ein ech-

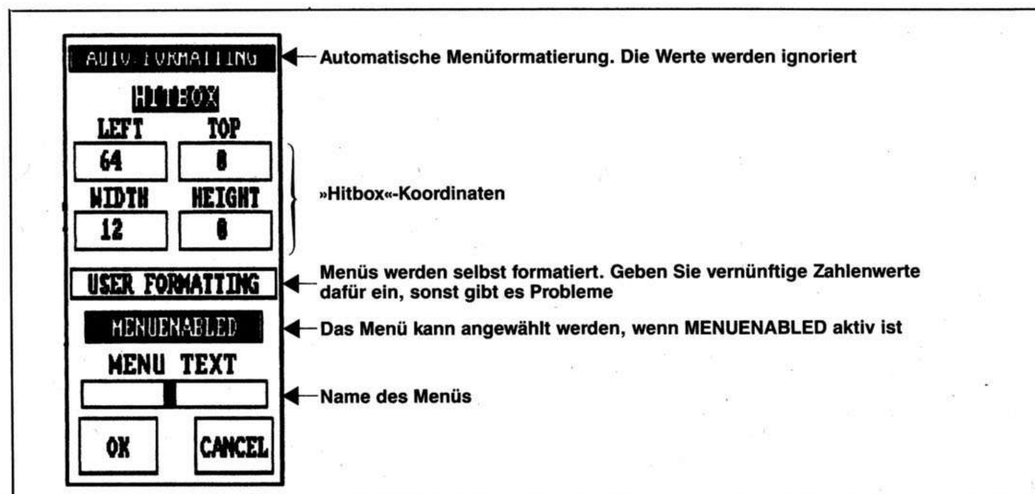


Bild 5. An diesem Requester brauchen Sie nichts einzustellen, wenn Sie Standard-Menüs erhalten wollen. Mit wenigen Mausklicks verändern Sie die Einstellungen.

tes Gadget in das Fenster eingesetzt. Wenn Sie wollen, können Sie sich jetzt den Quellcode für dieses Gadget unter dem Namen »gadget_1.src« erzeugen lassen und ihn mit »include "gadget_1.src"« in das Window-Anzeigeprogramm von Listing 2 einbauen. Sie sehen, daß zum Anzeigen von Gadgets das Programm nicht geändert werden muß.

Wenn Sie Ihr Gadget an eine andere Stelle des Fensters bewegen wollen, wählen Sie einfach »Move a Gadget«. Eine Dialogbox informiert darüber, daß man das gewünschte Gadget mit der Maus anklicken und verschieben soll. Schalten Sie also die Dialogbox mit »OK« ab und verschieben Sie das Gadget, wohin Sie wollen. Falls Sie mehrere Gadgets verwenden, achtet PowerWindows automatisch darauf, daß sie sich nicht überlappen – außer man schaltet das »Gadget collision checking« im Preferences-Menü ab, was aber nicht zu empfehlen ist! Intuition liefert nämlich unter Umständen unsinnige Informationen, wenn sich zwei Gadgets überlappen.

Zum Verändern der Gadget-Größe dient »Resize a Gadget«. Klicken Sie nach der Anwahl dieses Menüpunktes wieder auf das OK-Feld der Dialogbox und dann auf das Gadget, das Sie in der Größe verändern wollen. Den Mausknopf müssen Sie nicht festhalten. Stellen Sie die gewünschte Größe durch Bewegen der Maus ein und drücken Sie die linke Maustaste.

Beim Arbeiten mit Gadgets verwendet PowerWindows immer dieselbe Technik: Nach der Anwahl eines Gadget-Menüpunktes klicken Sie auf das Gadget, mit dem Sie arbeiten möchten – dazu werden Sie mit einer Dialogbox aufgefordert. Wenn Sie diese Boxen nach einiger Zeit nicht mehr sehen wollen, können Sie sie abschalten, indem Sie den Menüpunkt »OK prompts« im »Preferences«-Menü ausschalten. Mit <HELP> läßt sich die Dialogbox trotzdem auf Wunsch auf den Bildschirm holen, wenn Sie beispielsweise vergessen haben, welcher Befehl gerade gewählt wurde.

Sollte es Ihnen passieren, daß PowerWindows kein Menü mehr anzeigt, wartet das Programm höchstwahrscheinlich auf das Anklicken eines Gadgets. Drücken Sie dann <HELP>, um zu sehen, was das Programm von Ihnen erwartet.

Gadgets sind beweglich

Wie Sie sicher schon gemerkt haben, hat PowerWindows das erste Gadget automatisch als Boolean-Gadget definiert (man kann keine Zeichen eingeben oder Regler verschieben). Um sich die Sache genauer anzusehen, wählen Sie bitte »Define Gadget Type« und klicken auf das Gadget. Nun erscheint das Gadget-Steuerfeld (Bild 6).

Sie sehen, daß links oben »BOOLEAN« aktiviert ist, es handelt sich also – wie bereits festgestellt – um ein BOOLEAN-Gadget. Unten links stehen die Gadget-Flags; im Moment sind »GADHCOMP« und »RELVERIFY« aktiviert. GADGHCOMP bedeutet, daß das Gadget beim Anklicken invertiert werden soll. Mit GADGHBOX würde es beim Anklicken umrahmt werden, und GADGHNONE sorgt dafür, daß beim Anklicken überhaupt nichts passiert. PowerWindows zeigt allerdings alle Gadgets mit GADGHCOMP an; diese Flags werden erst im Quellcode gesetzt. RELVERIFY bedeutet ausgeschrieben »Release verify«, was auf deutsch »Bestätigung beim Loslassen« bedeutet. Intuition informiert das Programm, zu dem dieses Gadget gehört, erst dann über die Gadget-Betätigung, wenn der Mauszeiger über dem Gadget wieder losgelassen wurde. Das gibt dem Programmanwender zusätzliche Sicherheit, da er nach einem Klick auf das Gadget die Möglichkeit hat, seine Meinung zu ändern.

Wie stellt man vom Programm aus fest, daß ein Gadget angeklickt wurde? Wie Sie sicher schon geahnt haben, läuft das natürlich auch wieder mit dem IDCMP-System. Das Programm muß den »Class«-Wert der eintreffenden Intuition-Nachricht dann lediglich auf den Wert GADGETUP (Gadget losgelassen) oder GADGETDOWN (Gadget gedrückt) prüfen und das Ergebnis auswerten. Genau das macht Listing 6. Wenn Sie den Quellcode für das gerade definierte Gadget wieder unter dem Namen »gadget_1.src« abspeichern und ihn in das Programm einbinden (der »include«-Befehl steht bereits in Listing 6), sehen Sie folgendes: Sobald die Maustaste mit dem Zeiger über dem Gadget wieder losgelassen wurde, gibt es eine GADGETUP-Meldung.

Um auf das Anklicken eines Gadgets sofort reagieren zu können, sollten Sie statt RELVERIFY das Flag GADGIMMEDIATE (»Gadget sofort«) setzen. Dann bekommt das Programm sofort eine GADGETDOWN-Nachricht, wenn der Mausknopf mit dem Zeiger über dem Gadget gedrückt wurde. RELVERIFY und GADGIMMEDIATE lassen sich übrigens auch kombinieren – probieren Sie aus, was dann passiert!

Wenn Sie sich mit PowerWindows mehrere Boolean-Gadgets definieren, werden Sie feststellen, daß das Auswertungsprogramm zwar brav »GADGETDOWN« oder »GADGETUP« meldet, zwischen den einzelnen Gadgets aber offenbar nicht unterscheiden kann. Um sie unterscheiden zu können, muß jedes Gadget eine eigene Nummer bekommen. Diese Nummer geben Sie im Gadget-Steuerfeld unter »Gadget ID« ein. Sie ist ein Bestandteil der Gadget-Struktur und wird mit ihr im Quelltext abgelegt.

Um die Nummer des angeklickten Gadgets herauszufinden, muß der Intuition-Programmierer einen kleinen Umweg gehen. Intuition setzt nämlich in seine Gadget-Nachrich-

Ein BOOL-Gadget wird erzeugt
Ein P.ROP-Gadget wird erzeugt
Erzeugung eines STRING-Gadgets
String-Länge
Gadget-Koordinaten und Größe
»Nur-Rand«-Gadget. Siehe Text Seite 114
Zum Einbau von Grafiken
Gadget-Flags
von PW nicht benutzt
Bei Kickstart 1.3 unbenutzt
Gadget-ID
Name des Gadgets im Quelltext

Bild 6. Bei Gadgets haben Sie viele Einstellmöglichkeiten

Gadget-Listen

Wie Sie vielleicht wissen, erwartet Intuition alle Gadgets eines Fensters als verkettete Liste. Das heißt, daß in jeder Gadget-Struktur ein Zeiger auf das folgende Gadget steht (oder NULL, wenn es das letzte war). Manche Intuition-Funktionen arbeiten mit Teilen von Gadget-Listen. Es kann also unter Umständen notwendig sein, die Reihenfolge der Gadgets in der Liste festzulegen. Dazu dienen drei Menüpunkte des »Gadget«-Menüs:

- »Move gadget to start of list« bewegt das Gadget, das nun angeklickt wird, an den Anfang der Liste;
- »Move gadget to end of list« bewegt ein Gadget an das Ende der Liste;
- »Specify gadget successor« sagt dem Programm, daß »dieses« Gadget nach »jenem« in der Liste stehen soll. Für diesen Menüpunkt ist die Verwendung der »OK prompts« hilfreich.

ten nicht die Gadget-Nummer, sondern im Feld »IAddress« die Adresse des angeklickten Gadgets ein. Sie müssen also mit

```
GadgetAdr = Message ->
IAddress;
```

die Adresse auslesen. Nun wird mit

```
id = GadgetAdr ->
GadgetID
```

die vorher festgelegte Nummer abgeholt. Zusammen sieht das aus wie in Listing 7, das neben der Art der Gadget-Nachricht auch die Gadget-Nummer ausgibt. Übrigens: Wenn Sie bei »Gadget ID« im PowerWindows-Gadget-Steuerfeld nichts eingeben, hat das jeweilige Gadget die Nummer 0.

Die bisher verwendeten Boolean-Gadgets werden beim Anklicken komplementiert und gehen spätestens beim Loslassen der Maustaste wieder in den ursprünglichen Zustand zurück. Wenn Sie im Gadget-Steuerfeld das TOGGLESELECT-Flag setzen, wird das Gadget statt dessen beim Anklicken zwischen normalem und invertiertem Zustand umgeschaltet. Je nach der Einstellung von RELVERIFY und GADGIMMEDIATE bekommt das Programm bei jeder Umschaltung entweder GADGETUP, GADGETDOWN oder beides. Wenn Sie das Gadget bereits beim Öffnen des Fensters im angewählten Zustand haben möchten, müssen Sie das SELECTED-Flag setzen. Dieses Flag kann das Programm übrigens auch abfragen, beispielsweise so:

```
GadgetAdr = &Gadget;
if ((GadgetAdr->Flags &
SELECTED) == SELECTED)
printf ("Gadget ist
angewählt!\n");
else printf ("Gadget
nicht angewählt.\n");
```

Davor müssen Sie mit

```
struct Gadget *
GadgetAdr;
```

die Variable GadgetAdr deklarieren. Normalerweise numeriert PowerWindows die verwendeten Gadget-Strukturen im Quelltext durch. Wenn Sie ein Gadget später auf das SELECTED-Flag testen wollen, tragen Sie im SOURCE LABEL-Feld des Gadget-Steuerfeldes von PowerWindows den gewünschten Namen ein – so können Sie später darauf zugreifen.

Um Ihre Gadgets auch mit Text zu versehen, verwenden Sie den Menüpunkt »Work on Gadget Text« und klicken auf das gewünschte Gadget. Es erscheint ein neues Menü, das dieselben Textfunktionen wie der Textbefehl im »Current Window«-Menü enthält. Sie können Gadget-Text ebenfalls an jeder beliebigen Stelle des Fensters positionieren. Im Gegensatz zum Window-Text wird Gadget-Text aber automatisch mit dem Gadget angezeigt, wenn das

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <functions.h>
#include "gadget_1.src"
struct IntuitionBase *IntuitionBase;
main ()
{
  struct Window *MyWindow;
  struct IntuiMessage *Message;
  struct Gadget *address;
  int class,code,id;
  int MenuNumber;
  APTR Item;
  if (!(IntuitionBase = (struct IntuitionBase *)
          OpenLibrary ("intuition.library",0)))
     {printf ("Intuition nicht gefunden? Panik!\n"); exit (0
     );}
  MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
  re1);
  if (!MyWindow) {printf ("Fenster liess sich nicht öffnen!"
  );
                 CloseLibrary (IntuitionBase);
                 exit (0);}
  do {
   Wait (1<<MyWindow->UserPort->mp_SigBit);
    while (Message = GetMsg (MyWindow->UserPort))
     {
        class = Message -> Class;
        code = Message -> Code;
        address = Message -> IAddress;
        ReplyMsg (Message);
        id = address -> GadgetID;
        if (class == GADGETUP) printf ("Gadget %d up!\n",id)
        ;
        if (class == GADGETDOWN) printf ("Gadget %d down!\n"
        ,id);
        if (class == CLOSEWINDOW) break;
     }   /* of while */
  } while (class != CLOSEWINDOW);
  CloseWindow (MyWindow);
  CloseLibrary (IntuitionBase);
}
(C) 1988 M&T
```

Listing 6. Mit Hilfe der Flags »GADGETUP« und »GADGETDOWN« fragen Sie ab, ob der Anwender ein Gadget betätigt hat

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include "gadget_2.src"
struct IntuitionBase *IntuitionBase;
main ()
{
  struct Window *MyWindow;
  struct IntuiMessage *Message;
  struct Gadget *address;
  int class,code,id;
  int MenuNumber;
  APTR Item;
  if ((IntuitionBase = OpenLibrary ("intuition.library",0))
  == 0)
    {printf ("Intuition nicht gefunden? Panik!\n"); exit (0
    );}
  MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
  re1);
  if (!MyWindow) {printf ("Fenster liess sich nicht öffnen!"
  );
                 CloseLibrary (IntuitionBase);
                 exit (0);}
  ActivateGadget (&String,MyWindow,NULL); /* Gadget aktivier
  en */
  do {
   Wait (1<<MyWindow->UserPort->mp_SigBit);
    while (Message = GetMsg (MyWindow->UserPort))
     {
        class = Message -> Class;
        code = Message -> Code;
        address = Message -> IAddress;
        ReplyMsg (Message);
        id = address -> GadgetID;
        if (class == GADGETUP && id == 1)
          printf ("Es wurde '%s' eingegeben.\n",StringSIBuff)
          ;
        if (class == CLOSEWINDOW) break;
     }   /* of while */
   } while (class != CLOSEWINDOW);
  CloseWindow (MyWindow);
  CloseLibrary (IntuitionBase);
}
(C) 1988 M&T
```

Listing 7. String-Gadgets erfordern eine besondere Behandlung

Fenster geöffnet wird. Außerdem verschiebt sich Gadget-Text mit, wenn sein Gadget verschoben wird.

Auch Gadgets lassen sich mit Grafiken für den angeklickten und nicht angeklickten Zustand ausrüsten. Mit »Work on Gadget images« gelangen Sie in das Gadget-Grafikmenü. Vorher sollten Sie natürlich die gewünschte Grafik mit einem Malprogramm gezeichnet haben – speichern Sie das Bild in Form eines IFF-Pinsels mit der später benötigten Größe. Um ein Bild für den nicht angeklickten Zustand eines Gadgets zu laden, wählen Sie »Load gadget render imagery«; eine Grafik für den angeklickten Zustand lädt man mit »Load gadget select imagery«. Mit »Conform hitbox to render size« paßt PowerWindows die »Hitbox« des Gadgets, also den Bereich, in dem das Gadget auf einen Mausklick reagiert, auf die Größe der Grafik für den nicht angeklickten Zustand an. »Conform hitbox to render size« macht dasselbe für den angeklickten Zustand.

»Move gadget render imagery« und »Move gadget select imagery« erlauben es, für die Grafiken des nicht angeklickten beziehungsweise des angeklickten Zustands die Abstände von der linken oberen Ecke des Gadgets zu verändern. Sie sind normalerweise auf 0 eingestellt. Schließlich können Sie mit den beiden »Delete«-Funktionen bereits geladene Grafiken wieder löschen.

Gadgets klonen – kein Problem!

Sehr praktisch ist die »Clone a gadget«-Funktion. Damit können Sie ein völlig »baugleiches« Gadget herstellen, das Sie an einer beliebigen Stelle des Fensters positionieren. Neben der Größe des Gadgets werden auch alle Flags, Text etc. mitkopiert. Auf diese Weise kann man schnell umfangreiche Eingabefelder definieren, ohne immer wieder das gleiche eingeben zu müssen.

Normalerweise umrahmt PowerWindows die Gadgets automatisch mit einem Rechteck in der bei »Add Gadget« eingestellten Größe. Wenn Sie statt dessen Ihre eigene Umrahmung festlegen möchten, wählen Sie »Work on Gadget Borders« und klicken auf das gewünschte Gadget – Sie sind nicht auf rechteckige Rahmen beschränkt! Wie bei den »Work on...«-Funktionen üblich, erscheint auch hier ein neues Menü. Unter »Borders« stehen die Menüpunkte für die Erzeugung eines neuen Rahmens. Mit »Draw freehand border« können Sie einen Gadget-Rahmen »frei Hand« zeichnen. Sie klicken dazu mit der Maus auf den gewünschten Anfangspunkt der Umrandung, bewegen den Mauszeiger zum Ende der Linie, klicken auf diesen Punkt, bewegen den Zeiger zum Ende der zweiten Linie, klicken darauf. Diese Prozedur wiederholt sich, bis Sie mit einem Doppelklick abschließen. Das Wort »Rahmen« ist für diesen Linienzug eigentlich mißverständlich, da der Zug nicht geschlossen sein muß.

»Draw rectangular border« zeichnet ein Rechteck auf die übliche Weise (linke obere Ecke anklicken, Maus auf die rechte untere Ecke bewegen, klick); »Draw square border« zeichnet einen quadratischen Rahmen. Dafür müssen Sie mit der Maus in die Mitte des gewünschten Quadrates klicken und mit der Maus seine Größe einstellen. Um die Farbe, den Zeichenmodus oder den Abstand zwischen Rahmen und Gadget zu verändern, verwendet man »Edit gadget border«. Und mit »Delete gadget border« löscht man einen selbstgemachten Rand wieder.

Wichtig bei allen Border-Befehlen ist, daß die Gadget-»Hitbox« immer an derselben Stelle bleibt! Sie sollten das beim Bearbeiten der Gadget-Rahmen immer berücksichtigen.

Im »Other«-Menü gibt es einen besonderen Menüpunkt: »Standard Border«. Wenn Sie dort »Off« wählen, erzeugen Sie Gadgets ohne jeden Rand. Das geht aber nur, wenn Sie keinen eigenen Rahmen definiert hatten – der müßte eventuell vorher mit »Delete gadget border« wieder gelöscht werden.

Bisher wurden alle Gadget-Menüpunkte von PowerWindows an Boolean-Gadgets ausprobiert. Sie sind mit Sicherheit die meistbenutzten Gadgets – deshalb sind String- und Proportional-Gadgets aber nicht weniger wichtig. Um ein String-Gadget zu erzeugen, müssen Sie wie üblich mit »Add a gadget« die Größe festlegen. Nun aktivieren Sie mit »Define gadget type« das Gadget-Steuerfeld und klicken in der oberen Zeile auf »STRING«. Die maximale Länge des einzugebenden Strings müssen Sie bei »Length« auf den gewünschten Wert einstellen. Addieren Sie bitte zu diesem Wert noch eins hinzu, weil Intuition ein Byte des Strings als abschließendes Nullbyte braucht. Wenn der ganze String nicht in Ihr Gadget paßt, wird er von Intuition beim Eintippen gerollt. Werden aber mehr Zeichen eingegeben, als in »Length« angegeben, so blitzt der Bildschirm orange auf. Intuition kann bei der Eingabe den String entweder rechtsbündig oder zentriert schreiben – das stellen Sie mit »STRINGRIGHT« und »STRINGCENTER« ein.

Eingebaute Editierfunktion

Wenn Sie UNDOBUFFER einschalten, erzeugt PowerWindows automatisch einen Puffer für die UNDO-Funktion der String-Gadgets. Die kennen Sie nicht? Wenn Sie eine Veränderung in einem String-Gadget rückgängig machen wollen, probieren Sie es doch einmal mit <rechte-Amiga Q>. Hat das Gadget einen UNDO-Puffer, erscheint wieder die Version des Strings, die vor dem Anklicken des Gadgets existierte. Da ja immer nur ein String-Gadget den Puffer benötigt, erzeugt PowerWindows nur einen UNDO-Puffer. Er hat die Länge des längsten verwendeten String-Gadgets.

LONGINT macht das String-Gadget zu einem Zahlen-Gadget. Hier können also nur Ziffern eingegeben werden. Intuition rechnet die eingegebene Zahl automatisch in eine 32-Bit-Integerzahl um – wie man das im Programm auswertet, werden Sie gleich sehen. Bei der Eingabe der Zahl wird automatisch geprüft, ob sie sich in 32 Bits unterbringen läßt. Da diese Zahl maximal zehn Stellen plus Vorzeichen haben kann, ist eine größere Länge als zwölf für ein LONGINT-Gadget sinnlos.

Zuerst zur Abfrage eines »normalen« String-Gadgets. Definieren Sie dazu bitte ein String-Gadget mit den Flags UNDOBUFFER und STRINGCENTER mit der Gadget-ID 1 und dem Source-Label »String«. Die Länge des Strings

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <functions.h>
#include "gadget_prop.src"
struct IntuitionBase *IntuitionBase;
main ()
{
  struct Window *MyWindow;
  struct IntuiMessage *Message;
  struct Gadget *address;
  int class,code,id;
  int MenuNumber;
  APTR Item;
  if (!(IntuitionBase = (struct IntuitionBase *)
          OpenLibrary ("intuition.library",0)))
     {printf ("Intuition nicht gefunden? Panik!\n"); exit (0
     );}
  MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
  re1);
  if (!MyWindow) {printf ("Fenster liess sich nicht öffnen!"
  );
                 CloseLibrary (IntuitionBase);
                 exit (0);}
  do {
    Wait (1<<MyWindow->UserPort->mp_SigBit);
        while (Message = GetMsg (MyWindow->UserPort))
           {
             class = Message -> Class;
             code = Message -> Code;
             address = Message -> IAddress;
             ReplyMsg (Message);
             id = address -> GadgetID;
             if (class == GADGETUP && id == 2)
               {
                 printf ("Farbe = %d\n",PropGadSInfo.HorizP
                 ot/4096);
               }
             if (class == CLOSEWINDOW) break;
           }  /* of while */
     } while (class != CLOSEWINDOW);
  CloseWindow (MyWindow);
  CloseLibrary (IntuitionBase);
}
```


Listing 8. Proportional-Gadgets können Sie mit Hilfe der Werte in »PropGadSInfo« auswerten

können Sie beliebig wählen. Wichtig ist, daß Sie das Gadget-Flag RELVERIFY nicht ausschalten. Vergessen Sie bitte nicht, im Window-Steuerfeld für das Fenster, in dem das Gadget später erscheinen soll, GADGETUP, GADGETDOWN, CLOSEWINDOW, WINDOWCLOSE und ACTIVATE anzuklicken. Auf diese Weise erfährt unser Programm, wenn ein Gadget betätigt wurde oder das Fenster geschlossen werden soll – und das Fenster ist beim Start des Programms automatisch aktiv.

Wenn ein String-Gadget vom Benutzer verwendet wird, erfährt das Programm das erst, wenn nach der Eingabe <RETURN> gedrückt wurde – dann schickt Intuition eine GADGETUP-Meldung. Nun kann der eingegebene String ausgelesen werden. Jedes String-Gadget bekommt von PowerWindows automatisch einen Speicherbereich zugewiesen, in dem Intuition den String ablegen kann. Dieser Puffer hat den Namen »<Gadget-Name>SIBuff«, in diesem Fall also »StringSIBuff«. Mit einem einfachen »printf« holen Sie den String nun auf den Bildschirm. In Listing 8 funktioniert die Abfrage nach diesem Muster. Sobald das Programm eine GADGETUP-Meldung bekommt, prüft es auf die Gadget-ID 1 (die Sie vorher dem String-Gadget zugewiesen hatten) und druckt den eingegebenen String aus. Wenn Sie ein String-Gadget mit einem String »vorbesetzen« wollen, müssen Sie den jeweiligen Speicherbereich vor dem Öffnen des Fensters, in dem das Gadget steht, initialisieren, zum Beispiel so:

```
strcpy (StringSIBuff,
"Dieser Text steht schon
im Gadget!");
```

Wollen Sie in ein String-Gadget etwas hineinschreiben, wenn das Fenster schon offen ist, so müssen Sie den Text in den entsprechenden Puffer schreiben und

```
RefreshGadgets (gadgets,
window,requester)
```

aufrufen. Dabei enthält »gadgets« die Adresse des ersten Gadgets (bei PowerWindows-Gadgets normalerweise »Gadget1«), »window« die Adresse der Window-Struktur, in deren Fenster das Gadget steht, und »requester« die Adresse der Requester-Struktur, wenn die Gadgets in einem Requester stehen. Wenn nicht, muß für requester »NULL« eingesetzt werden.

Im Listing 8 wurde außerdem die Funktion

```
ActivateGadget (gadget,
window,requester)
```

verwendet. Diese Funktion erspart das Anklicken von String-Gadgets – man kann also in dem String-Gadget, das damit aktiviert wurde, sofort lostippen. Die Parameter der Funktion sind denen von »RefreshGadgets« sehr ähnlich, allerdings muß für »gadget« die Adresse des zu aktivierenden String-Gadgets angegeben sein. Für diese Funktion ist es besonders praktisch, wenn Sie Ihren verschiedenen String-Gadgets im Gadget-Steuerfeld von PowerWindows eigene Namen geben, da man so die Gadget-Adresse problemlos angeben kann. Das Fenster, in dem das Gadget aktiviert werden soll, muß zu diesem Zeitpunkt ebenfalls aktiv sein! Deshalb sollten Sie das ACTIVEWINDOW-Flag des Fensters auf jeden Fall setzen. Eine andere Möglichkeit ist, mit

```
ActivateWindow (window)
```

das Fenster zu aktivieren, wobei »window« ein Zeiger auf die Window-Struktur des zu aktivierenden Fensters ist.

Um die eingegebene Zahl bei einem LONGINT-Gadget abzufragen, müssen Sie das »LongInt«-Feld der zu diesem Gadget gehörenden »StringInfo«-Struktur auslesen. Diese Strukturen haben bei PowerWindows den Namen <Gadget-Name>SInfo. Haben Sie also Ihr Nummern-Eingabe-Gadget »Zahl« genannt, erfahren Sie mit

```
Eingabe = ZahlSInfo.
LongInt
```

die eingegebene Zahl.

Geregelte Schiebung

Für die intuitive Benutzeroberfläche des Amiga sind Proportional-Gadgets sehr wichtig, weil man sich unter so einem »Schieberegler« sofort etwas vorstellen kann. Proportional-Gadgets (im folgenden Prop-Gadgets genannt) bieten vielerlei Möglichkeiten: Der »Knopf« des Schiebereglers kann entweder waagerecht, senkrecht oder in beiden Richtungen verschoben werden – je nachdem, wie das Programm es vorsieht. Die Größe des Knopfes innerhalb des Schiebereglers ist beliebig festlegbar und für den Benutzer leicht umzusetzen: Füllt beispielsweise der Knopf eines Scroll-Reglers bei einem Textprogramm ein Drittel des Reglers aus, so weiß man sofort, daß gerade ein Drittel des Textes angezeigt wird. Wie Sie sicher wissen, kann man Prop-Gadgets auf zwei Arten anklicken: Entweder man klickt auf den Knopf und verschiebt ihn beliebig weit; oder man klickt (bei einem waagerechten Regler) rechts oder links des Knopfes, um ihn um einen Schritt zu verschieben.

Das Erzeugen von Prop-Gadgets ist mit PowerWindows sehr einfach. Legen Sie wie üblich unter »Add Gadget« die gewünschte Größe fest und aktivieren Sie das Gadget-Steuerfeld für Ihr Gadget. In der obersten Zeile klicken Sie auf »PROPORTIONAL«. Unter diesem Wort finden Sie die Felder »HBODY« und »VBODY«; sie enthalten beide eine Null. In diesen beiden Feldern wird die horizontale beziehungsweise vertikale Schrittweite eingetragen, die das Gadget haben soll. Bei jedem Klick auf das Prop-Gadget neben dem Knopf wird der Knopf um diesen Wert verschoben; gleichzeitig gibt der Schrittweiten-Wert die Größe des Knopfes an. Wenn Sie zur Farbeinstellung ein Gadget mit 16 Schritten definieren, hat der Gadget-Knopf ein Sechzehntel der Gadget-Größe, und der gesamte Regler ist in 16 Schritte aufgeteilt.

Etwas weiter unten sehen Sie die Flags »FREEHORIZ«, »FREEVERT« und »PROPBORDERLESS«. FREEHORIZ und FREEVERT geben den Gadget-Knopf sozusagen in waagerechter und senkrechter Richtung »frei«. Sie müssen also mindestens eines dieser beiden Flags setzen und entsprechend einen Schrittweiten-Wert eintragen. Sind FREEHORIZ und FREEVERT gesetzt, so kann sich der Gadget-Knopf innerhalb des Gadgets frei bewegen. PROPBORDERLESS sollten Sie nur setzen, wenn Sie nicht möchten, daß Intuition dieses Gadget automatisch mit einem Rand versieht. Erzeugen Sie also ein Prop-Gadget mit FREEHORIZ und HBODY 16. Geben Sie ihm die ID 1 und den Namen PropGad. Übrigens: »Work on Gadget Borders« funktioniert bei Prop-Gadgets nicht, da diese keinen selbstgemachten Rahmen haben.

Haben Sie alles eingestellt? Dann verlassen Sie das Gadget-Steuerfeld und probieren das Prop-Gadget aus. Experimentieren Sie mit den Schrittweiten und Flags, um Erfahrungen damit zu sammeln. Wenn Sie den Quelltext für ein Prop-Gadget speichern (unter dem Namen »gadget_prop«), vermerkt PowerWindows die aktuelle Stellung des Knopfes als Startposition.

Für die Abfrage eines Prop-Gadgets sind zwei Variablen sehr wichtig: HorizPot und VertPot. In diesen beiden Variablen steht die horizontale beziehungsweise vertikale Position des Schiebereglers. Sie stehen in der »PropInfo«-Struktur jedes Prop-Gadgets. Diese Struktur bekommt von PowerWindows den Namen <Gadget-Name>SInfo – im Beispiel also »PropGadSInfo«.

Um die Stellung des Gadgets im Programm richtig abfragen zu können, müssen Sie wissen, daß Intuition die Prop-Gadget-Werte grundsätzlich als 16-Bit-Integerzahlen verwaltet. Steht der Regler (bei einem horizontalen Prop-Gadget) ganz links, so ist HorizPot 0; steht er am rechten Anschlag, ist HorizPot 65535. Haben Sie also ein Gadget mit der Schrittweite 16 definiert, müssen Sie den entsprechenden Wert durch (65536/16), also durch 4096, teilen, um Werte zwischen 0 und 15 zu erhalten. Im Listing 10 ist das mit dem vorher definierten Prop-Gadget ausgeführt.

Die Einstellung des Schiebers

Zwei weitere Variablen in der PropInfo-Struktur sind ebenfalls wichtig: HorizBody und VertBody. Sie geben die Größe des Gadget-Knopfes im Verhältnis zum gesamten Gadget an. Es handelt sich ebenfalls um 16-Bit-Variablen: Bei einem Wert von 65535 wäre also das gesamte Gadget ausgefüllt. Um aus einer gewünschten Schrittweite auf den HorizBody- beziehungsweise VertBody-Wert zu kommen, müssen Sie einfach 65536 durch die Schrittweite dividieren – PowerWindows macht das automatisch. Interessant wird diese Tatsache, wenn Sie die Werte eines Prop-Gadgets in einem Programm verändern wollen. Dazu dient die Funktion

```
ModifyProp (gadget,
window, requester,
flags, horizPot,
vertPot, horizBody,
vertBody);
```

»gadget«, »window« und »requester« dürften inzwischen schon bekannt sein. »flags« enthält die Flags für das Proportional-Gadget. Wenn Sie sie nicht verändern wollen, setzen Sie dafür einfach <GadgetName>SInfo.Flags ein. Wollen Sie et-

was verändern, denken Sie daran, das AUTOKNOB-Flag zu setzen, weil sonst der Gadget-Knopf spurlos verschwindet! In »horizPot«, »vertPot«, »horizBody« und »vertBody« werden die neuen Werte für diese Variablen eingetragen.

Gerade bei Prop-Gadgets ist es oft notwendig, daß sich die Größe des Gadgets mit der Größe des Windows, in dem es sich befindet, verändert oder daß das Gadget an den Fensterkanten »klebt«. Ein Beispiel dafür wäre der Scrollbalken eines Textprogramms, der sich den Größenveränderungen des Textfensters anpassen muß. Zu diesem Zweck dienen die Gadget-Flags »GRELBOTTOM«, »GRELRIGHT«, »GRELWIDTH« und »GRELHEIGHT«. »GREL« bedeutet »Gadget relativ« – relativ in diesem Fall zur Größe des Fensters. Sind GRELWIDTH oder GRELHEIGHT gesetzt, so paßt Intuition die Gadgetbreite beziehungsweise -höhe bei jeder Änderung der Fenstergröße an. GRELBOTTOM und GRELRIGHT halten den Abstand des Gadgets zur unteren und rechten Fensterkante konstant. Wenn Sie mit diesen Flags unter PowerWindows arbeiten, sollten Sie die Größenverhältnisse und Abstände nach Ihren Wünschen einstellen und die gewünschten Flags setzen. Die Flags wirken sich aber nicht bei der Arbeit mit PowerWindows, sondern erst im eigenen Programm aus. Die Größenverhältnisse bleiben bei jeder Einstellung des Fensters erhalten. Wird dieses System auf ein Prop-Gadget angewandt, paßt Intuition die Knopfgröße immer auf die aktuellen Größenverhältnisse an. Die GREL-Flags beherrschen Sie am leichtesten durch Ausprobieren – hier ist Experimentieren wirklich der beste Weg, um mit der Materie zurechtzukommen. Im allgemeinen kann man sagen, daß es meist sinnlos ist, mehr als zwei GREL-Flags gleichzeitig zu setzen.

Um einen eigenen Knopf für ein Prop-Gadget zu erzeugen, verwenden Sie einfach »Work on gadget images« für das Prop-Gadget.

Gadgets kann man ebenso wie Menüs inaktiv machen, so daß sie nicht mehr angewählt werden können. Wenn ein Gadget von Anfang an abgeschaltet sein soll, schalten Sie sein »GADGDISABLED«-Flag ein. Leider ist diese Funktion ebenfalls erst im späteren Programm zu sehen – noch nicht während der Arbeit mit PowerWindows.

Um Gadgets im Programm ein- und auszuschalten, verwenden Sie

```
OnGadget (gadget,window,
requester)
```

beziehungsweise

```
OffGadget (gadget,
window,requester)
```

Die Parameter sind dieselben wie bei den anderen Gadget-Funktionen.

PowerWindows stellt sogar zwei »Gadget-Typen« zur Verfügung, die Intuition eigentlich gar nicht kennt. Es handelt sich dabei um die »BORDER ONLY«- und die »IMAGE ONLY«-Gadgets. Streng genommen sind das natürlich keine neuen Gadget-Typen, sondern sinnvolle Anwendungen der vorhandenen. BORDER ONLY dient dazu, beliebige Linienzüge in ein Fenster zu zeichnen, die Gruppen von Gadgets umrahmen. Dazu definieren Sie mit »Add Gadget« ein normales Gadget (wenn Sie andere Gadgets umrahmen wollen, müssen Sie das »Gadget collision checking« im Preferences-Menü abschalten). Wenn Sie wollen, können Sie den Rahmen natürlich mit »Work on gadget borders...« noch verändern und beispielsweise durch einen Linienzug ersetzen. Wenn Sie nun im Gadget-Steuerfeld BORDER ONLY anklicken, schreibt PowerWindows lediglich eine Border-Struktur für den gerade definierten Rahmen oder Linienzug in den Quelltext – auch wenn es während der PowerWindows-Benutzung so scheint, als befinde sich an dieser Stelle ein Gadget. Wenn Sie mehrere BORDER ONLY-Gadgets verwenden, wird im Quelltext eine verkettete Liste von Border-Strukturen erzeugt. Damit diese automatisch beim Öffnen des Fensters angezeigt werden, klicken Sie im Quelltext-Steuerfeld (Bild 3) auf »Borders/Images Gadget«. Nun erzeugt PowerWindows ein unsichtbares Mini-Gadget, das als Verbindung zwischen Border-Struktur und Window dient.

Requester leichtgemacht

Dasselbe Verfahren gibt es bei IMAGE ONLY auch für Grafiken, die Sie beliebig in Ihrem Fenster verteilen können. Dazu müssen Sie ein Gadget erzeugen und es mit »Work on gadget images« mit einer Grafik versehen. Nun aktivieren Sie wieder »Borders/Images Gadget«. PowerWindows bringt dann wie bei BORDERS ONLY die als IMAGE ONLY definierten Grafiken auf den Schirm. Die Einstellung der Gadget-Flags spielt bei BORDER ONLY und IMAGE ONLY-Gadgets keine Rolle – sie wird von PowerWindows automatisch vorgenommen. Bei diesen beiden »Gadget-Typen« wird Gadget-Text nicht angezeigt. Wenn Sie für den angeklickten Zustand Ihres IMAGE ONLY-Gadgets eine Grafik laden, wird sie beim Schreiben des Quelltextes nicht berücksichtigt.

Ein Requester ist ein spezielles Eingabefeld, das vom Programm aus mit »Request (requester)« aufgerufen wird. Er kann wie ein Fenster Gadgets, Texte und Grafiken enthalten, aber keine Menüs. Die Besonderheit beim Requester ist, daß er jede Eingabe an das Fenster, von dem er aufgerufen wurde, blockiert. Ein Requester verschwindet automatisch vom Bildschirm, sobald ein Gadget angeklickt wird, das als »Ende-Gadget« markiert ist. Requester sollten Sie dann benutzen, wenn Sie eine Reaktion des Programmanwenders erzwingen wollen – er muß zumindest auf »Ende« oder »Cancel« klicken, um den Requester wieder verschwinden zu lassen.

PowerWindows unterstützt Requester, aber leider nicht sehr komfortabel. Im Window-Steuerfeld befindet sich links unten das Feld »TYPE«. Wenn Sie auf das danebenstehende »WINDOW« klicken, verwandelt es sich in »REQUESTER«. Nun erzeugt PowerWindows statt einem Fenster einen Requester.

Ein Requester wird immer in Zusammenhang mit einem Fenster aufgerufen – alle Eingaben (Gadget-Nachrichten) des Requesters werden an dieses Fenster geschickt. Gleichzeitig wird die Eingabe des Fensters blockiert. Außerdem muß der Requester in diesem Fenster Platz haben. Um mit PowerWindows einen zu erzeugen, verwenden Sie zunächst wieder »Open a new window«. Geben Sie diesem Fenster die zusätzlichen Flags CLOSEWINDOW, WINDOWCLOSE und ACTIVATE. Das Fenster sollte einen großen Teil des Bildschirms ausfüllen, um später den Requester aufzunehmen. Nun definieren Sie ein zweites Fenster, das die Umrisse des späteren Requesters angibt. Beachten Sie bei der Positionierung dieses Fensters, daß der Requester später so weit von dem Fenster, mit dem er aufgerufen wird, entfernt ist, wie jetzt von der linken oberen Bildschirmecke. Diese Tatsache erfordert einiges Probieren, um damit vertraut zu werden.

Aktivieren Sie nun das Window-Steuerfeld und schalten Sie TYPE für das neue Fenster auf REQUESTER um. Sie sollten sämtliche Window-Flags (auf der rechten Seite des Feldes) außer SMART_REFRESH ausschalten. Jeder Requester benötigt mindestens ein Gad-

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <functions.h>
#include "requester.src"
struct IntuitionBase *IntuitionBase;
main ()
{
  struct Window *MyWindow;
  struct IntuiMessage *Message;
  struct Gadget *address;
  int class,code,id;
  int MenuNumber;
  APTR Item;
  if (!(IntuitionBase = (struct IntuitionBase *)
        OpenLibrary ("intuition.library",0)))
     {printf ("Intuition nicht gefunden? Was ist los?\n"); e
     xit (0);}
  MyWindow = (struct Window *) OpenWindow (&NewWindowStructu
  re1);
  if (!MyWindow) {printf ("Fenster liess sich nicht öffnen!"
  );
                    CloseLibrary (IntuitionBase);
                    exit (0);}
  Request (&RequesterStructure2,MyWindow);
  Wait (1<<MyWindow->UserPort->mp_SigBit);
  CloseWindow (MyWindow);
  CloseLibrary (IntuitionBase);
}
(C) 1988 M&T
```

Listing 9. Auch Requester verwalten Sie mit PowerWindows sehr einfach

get, mit dem er beendet werden kann, sonst werden Sie ihn nie mehr los. Bauen Sie also ein Boolean-Gadget in den Requester ein, geben Sie ihm den Text »Ende.« und klicken Sie im Gadget-Steuerfeld für dieses Gadget auf ENDGADGET. Dieses Flag ist sehr wichtig: Es sagt Intuition, daß es den Requester wieder vom Bildschirm nehmen soll. Um einen Requester auf den Schirm zu bringen, verwendet man die Funktion

```
Request (Requester,
Window)
```

»Requester« ist ein Zeiger auf eine Requester-Struktur, in diesem Fall »&RequesterStructure2«. »Window« ist der Zeiger auf die Window-Struktur des Fensters, zu dem der Requester gehört.

In Listing 10 wird der soeben definierte Requester auf den Bildschirm gebracht, wozu Sie vorher mit PowerWindows den Quellcode als »Requester.src« abspeichern müssen. Intuition entfernt den Requester automatisch, sobald »Ende.« angeklickt wird. Ein Klick auf das Schließ-Gadget des Fensters beendet das Programm. Wie Sie sehen, hat der Requester eine weiße Hintergrundfarbe (falls Sie mit den üblichen Workbench-Farben arbeiten). Diese Farbe kann man im »BLOCK«-Feld am rechten Rand des Window-Steuerfeldes ändern. Wenn der Requester die Hintergrundfarbe 0 hat, sollten Sie ihn mit Hilfe eines BORDER ONLY-Gadgets umrahmen.

Eine besondere Form von Requestern sind »Double-Menu«-Requester. Sie werden durch einen Doppelklick auf die rechte Maustaste aufgerufen. Dazu definieren Sie mit PowerWindows einen normalen Requester und rufen im Programm die Funktion

```
SetDMRequest (Window,
Requester)
```

auf. Die Parameter dieser Funktion sind gegenüber der Request-Funktion vertauscht – eine beliebte Fehlerquelle. Bei Double-Menu-Requestern gibt es eine Besonderheit: Man muß sie mit

```
EndRequest (Requester,
Window)
```

selbst wieder »vom Bildschirm holen«. Dazu sollten Sie auf eine Gadget-Nachricht aus diesem Requester warten und ihn dann entfernen. Mit

```
ClearDMRequest (Window,
Requester)
```

schaltet man einen Double-Menu-Requester wieder ab, so daß er nicht mehr per Doppelklick gewählt werden kann.

Wir sind nun am Schluß des ersten Workshops im AMIGA-Sonderheft angelangt. Hoffentlich können Sie die Vorteile von PowerWindows in Zukunft noch besser nutzen. Wenn es eine Meisterschaft im Fensterln gäbe, würden Sie mit den erworbenen Kenntnissen jetzt bestimmt ganz vorn in der Rangliste liegen. *(Andreas Lietz/so)*

Dem CLI das Handwerk gelegt

Mit dem CLI ist den Entwicklern des Amiga kein Meisterwerk gelungen. Spartanische Editiermöglichkeiten, schlechte Benutzerführung und fehlende Mausunterstützung lassen den Workbench-verwöhnten Anwender schnell verzweifeln. »ExCLI« bringt Ihnen den lange vermißten Komfort.

Computer gelten nicht zu Unrecht als fleißig aber dumm. Leider scheinen sie allzu oft diese Eigenschaften auch beim Benutzer zu erwarten. Gehören Sie zu den geduldigen Zeitgenossen, die nach jedem kleinen Tippfehler im CLI schuldbewußt die Fehlermeldung entgegennehmen und von vorne zu tippen anfangen? Oder revoltieren Sie gelegentlich mit einem kräftigen Fluch gegen die Arroganz des Systems, das Ihnen diese endlosen Tippübungen auferlegt?

Wenn Sie Ihr CLI auch manchmal verwünschen, sollten Sie sich »ExCLI« näher ansehen:

ExCLI ist eine neue Oberfläche, die dem CLI »die Hände bindet« und dessen Aufgaben übernimmt. Nach dem Start haben Sie erweiterte Editiermöglichkeiten. Ausgeführte Befehle können wieder auf den Bildschirm geholt und – eventuell nach Korrektur – erneut gestartet werden (»History«-Funktion). Eingebaute Befehle beschleunigen häufig benutzte CLI-Operationen wie »cd«, »dir« oder »type«. Sie können sich beliebige Zeichenfolgen als Kommando definieren, die auf einen Tastendruck hin ausgeführt werden. <HELP> bringt jederzeit eine Hilfsseite mit den wichtigsten Befehlen und der aktuellen Funktionstastenbelegung auf den Bildschirm.

Haben wir Ihnen den Mund wässrig gemacht? Dann sollten Sie »ExCLI.c« (Listing 1) abtippen und mit Aztec-C übersetzen. Die Aufrufe lauten:

```
cc ExCLI +1
ln ExCLI -lc32
```

Wenn Sie sich das Abtippen ersparen wollen, finden Sie auf der Programmservicediskette eine lauffähige Version einschließlich Sourcecode. Bevor Sie ExCLI starten, stellen Sie sicher, daß einige Voraussetzungen erfüllt sind: ExCLI richtet ein logisches Laufwerk mit Namen »C« in der RAM-Disk ein. Wenn Sie Ihre gewohnten CLI-Befehle benutzen wollen, sollten Sie diese entweder vor dem Start von ExCLI in dieses Verzeichnis kopieren oder die Zeile

```
Path SYS:c add
```

eingeben. Damit ExCLI korrekt arbeitet, müssen folgende Files auf der Startdiskette sein:
- »c/assign«
- »c/copy«
- »c/run«
- »l/RamHandler«

Wenn Sie sicher sind, daß diese Voraussetzungen erfüllt sind, starten Sie ExCLI entweder durch Doppelklick auf das Icon oder vom CLI aus mit

```
excli
```

gefolgt von <RETURN>. Das Programm richtet – soweit noch nicht vorhanden – eine RAM-Disk mit dem Unterverzeichnis »C« ein. Die Files »assign« und »run« werden in dieses Verzeichnis kopiert. Darauf wird es per »assign« zum logischen Laufwerk »C:«. Wenn Sie alle benötigten Befehle, die in ExCLI nicht enthalten sind, in dieses »Laufwerk« kopieren, brauchen Sie nicht ständig die Systemdiskette einlegen und können bequem mit einem Laufwerk arbeiten.

Natürlich können Sie diese Operationen auch mit einem sogenannten »Batchfile« vereinfachen. Sehen Sie sich dazu unser Beispiel »ExCLI.bat« an (Listing 2). ExCLI sucht im aktuellen Verzeichnis nach der Datei »ExCLI.key«. In dieser sind die zuletzt gespeicherten Tastaturbelegungen enthalten. Ist dieses File vorhanden, so wird diese Belegung geladen. Wenn nicht, startet ExCLI mit der Standard-Belegung. Zuletzt richtet sich ExCLI noch das CLI-Fenster für seine Zwecke ein. Dadurch sparen Sie Speicherplatz, denn Sie brauchen für ExCLI kein eigenes Fenster.

Beschleunigung durch Einbau-Befehle

ExCLI hat 13 eingebaute Kommandos. Diese müssen nicht geladen werden, wodurch die Arbeit beschleunigt wird. ExCLI unterscheidet bei Befehlen zwischen Groß- und Kleinschreibung. Alle eingebauten Befehle sind in Kleinbuchstaben. Bei gleichlautenden CLI-Befehlen (beispielsweise »dir« und »DIR«) können Sie einen beliebigen Buchstaben als Großbuchstaben angeben, um das CLI-Äquivalent zu starten.

Die neuen Befehle im Überblick:

dir listet Directory-Einträge auf ähnliche Weise wie das entsprechende CLI-Kommando. Unterverzeichnisse werden farblich abgehoben, die Größe der Files angezeigt.

cd legt das aktuelle Verzeichnis fest.

pfad zeigt alle Suchpfade an.

del löscht ein angegebenes File.

mdir legt ein neues Unterverzeichnis an.

ren benennt einen Eintrag um.

type gibt den Inhalt eines Files auf den Bildschirm aus. Das File wird im Gegensatz zum CLI-Type seitenweise gelistet und eventuell enthaltene Steuerzeichen werden unterdrückt, so daß beim Listen von Binärdateien der Bildschirminhalt nicht zerstört wird.

help zeigt eine Hilfsseite an. Diese enthält die eingebauten Befehle sowie die aktuelle Belegung der Funktionstasten.

cls löscht den Fensterinhalt und setzt das Prompt in die linke obere Ecke.

save speichert die aktuelle Tastenbelegung im File »ExCLI.key«. Wenn Sie vermeiden wollen, daß im aktuellen Verzeichnis gespeichert wird, müssen Sie das Verzeichnis als Argument übergeben.

view Belegung einer Taste feststellen:

»view A« zeigt die Belegung von <SHIFT a>.

new löscht eine Tastenbelegung.

exit beendet ExCLI.

Daneben können Sie natürlich weiterhin alle Befehle verwenden, die im logischen Laufwerk »C:« liegen oder in einem der mit einem Suchpfad belegten Verzeichnisse.

Ein großer Nachteil des CLI sind die fehlenden Editiermöglichkeiten für die Kommandozeile. Ein falsch eingegebener Befehl muß bis zum Fehler mit <BACKSPACE> gelöscht werden. Hatten Sie bereits <RETURN> gedrückt, so gibt es keine andere Möglichkeit als die Zeile neu zu tippen. Bei Kopierbefehlen oder Compileraufrufen mit langen Optionslisten kann dies sehr ärgerlich werden. Mit Ex-CLI haben Sie Möglichkeiten, die Kommandozeile zu editieren:

<**BACKSPACE**> hat die gleiche Funktion wie im CLI, es löscht das Zeichen links vom Cursor.

<**DEL**> löscht das Zeichen unter dem Cursor.

<**CRSR-links**> bewegt den Cursor ein Zeichen nach links.

<**CRSR-rechts**> bewegt den Cursor ein Zeichen nach rechts.

<**SHIFT CRSR-links**> bewegt den Cursor an das linke Ende der Eingabezeile.

<**SHIFT CRSR-rechts**> bewegt den Cursor an das rechte Ende der Eingabezeile.

<**CRSR-up**> fügt ein Zeichen an der Cursorposition ein (ExCLI arbeitet im Overwrite-Modus, daher müssen Sie für Zeichen, die eingefügt werden sollen, vorher Platz reservieren).

<**CRSR-down**> löscht das Zeichen unter dem Cursor (gleiche Funktion wie <DEL>).

<**SHIFT CRSR-up**> löscht die Eingabezeile rechts von der Cursorposition.

<**SHIFT CRSR-down**> löscht die Eingabezeile.

Jeder ausgeführte Befehl wird mit der »History«-Funktion zwischengespeichert. Sie können jederzeit die letzten zehn Befehle mit einem Tastendruck wieder auf dem Bildschirm erscheinen lassen, eventuell editieren und erneut ausführen.

Der zuletzt ausgeführte Befehl liegt immer auf <F10>, sobald ein neuer Befehl auf <F10> liegt, wird die gesamte History um eine Funktionstaste nach links »kopiert«. Der »älteste« Befehl fällt dabei heraus. Um sich die Wirkungsweise dieser History-Funktion zu verdeutlichen, lassen Sie einige Befehle ausführen. Drücken Sie nach jedem Befehl auf <HELP>. Sie erhalten jedesmal die aktuelle Belegung.

Eine besondere Funktion hat die Taste <TAB>. Mit ihr wird der Schreibmaschinenmodus eingeleitet. Sie können jetzt eine beliebige Zeile editieren. Sobald Sie <RETURN> drücken, wird diese an den Drucker geschickt. Sie können bei ExCLI bis zu 80 Zeichen pro Zeile editieren. Kurz vor diesem Limit werden Sie durch einen Bildschirmblitz bei jedem Tastendruck gewarnt.

Mit ExCLI können Sie bis zu 95 Tasten mit eigenen Shortcuts belegen. Sie legen eine beliebige Zeichenfolge (bis zu einer Länge von 80 Zeichen) auf eine einzige Taste. Dazu schreiben Sie die gewünschte Zeichenfolge in die Eingabezeile, drücken aber nicht <RETURN>, sondern zweimal <ESC>. Jetzt drücken Sie eine beliebige Taste, mit der Sie diese Zeichenfolge abrufbar machen wollen.

Das Abrufen funktioniert auf ähnliche Weise: Drücken Sie einmal <ESC>. Geben Sie eine beliebige Taste ein. Wenn diese belegt war, wird der zugehörige Befehl ausgeführt.

Sie können die Tasten mit den ASCII-Codes von 32 bis 126 für Shortcuts benutzen. Diese umfassen alle Buchstaben, Ziffern und Sonderzeichen.

Vermutlich wollen Sie nicht bei jedem Start von ExCLI alle Shortcuts neu definieren. Mit dem »save«-Befehl speichern Sie die Belegung. Dafür wird das File »ExCLI.key« im aktuellen Verzeichnis verwendet. Wenn Sie in ein anderes Verzeichnis schreiben wollen, geben Sie dieses als Parameter an.

Sie beenden ExCLI mit dem Befehl »exit« oder mit der Tastenkombination <CTRL d>. Dies sollten Sie erledigen, bevor Sie mit »endcli« das CLI beenden, da sonst Ihr CLI »arbeitsunfähig« wird: Es erhält den Befehl zum Beenden, hat aber noch einen Task gestartet, der nicht beendet ist (ExCLI). Der Amiga wartet auf das Beenden dieses Tasks, bevor er das Kommando ausführt. Gleichzeitig wartet der Task auf die Ausführung des Kommandos bevor er sich verabschiedet – ein eindrucksvolles Demo der Multitasking-Schwierigkeiten.

Erweiterungen

ExCLI ist eine komfortable Oberfläche für das CLI. Sicher treffen die gebotenen Möglichkeiten nicht jedermanns Geschmack. Wenn Sie das Listing studieren, sollten Sie in der Lage sein, eigene Erweiterungen anzubringen. Mausfans möchten vielleicht Menüs und Gadgets einrichten. Hierfür finden Sie eine Anregung im C-Kurs auf Seite 61. Wenn Sie die Cursortasten mit anderen Funktionen belegen wollen, sollten Sie die Zeilen 545 bis 592 im Listing aufmerksam lesen. Natürlich ist es auch möglich, die Funktionstasten fest zu belegen, in Zeile 393 bis 402 finden Sie dazu Informationen.

In jedem Fall erleichtern Sie sich mit ExCLI enorm die Arbeit mit dem CLI. *(Arno Gölzer/so)*

Programmname:	ExCLI
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec
Aufrufe:	siehe Text

```
Programmautor: Arno Gölzer
--------------------------

 1 ALO #include <stdio.h>
 2 RC  #include <sgtty.h>
 3 1o  #include <libraries/dos.h>
 4 m3  /* Maximale Anzahl Zeichen pro Zeile; Anzahl der ExCLI-Befe
       hle */
 5 Il  #define MAXC 80
 6 Hh  #define KOM  14
 7 b0  /* Festlegen der globalen Variablen */
 8 BV  FILE *tastatur, *file, *drucker;
 9 rL  int dir(),cd(),del(),mdir(),help(),cls(),ren();       /* ei
       gene KOMmandos */
10 76  int ende(),view(),type(),save(),new(),pfad();
11 7D  int maschine=0;
12 S1  int flag=0;
13 LM  int p=-1;
14 Kh  int z=-1;
15 gQ  unsigned char aktuellDir[200];
16 Qc  unsigned char store[200];
17 pn  unsigned char zeile[MAXC+1];
18 hD  unsigned char kommando[MAXC+1];
19 bE  unsigned char parameter[MAXC+1];
20 Rp  unsigned char cbef[MAXC+7];
21 65  unsigned char esc[96][MAXC+1];
22 5p  unsigned char ftaste[10][MAXC+1];
23 rL  unsigned char par[2][MAXC+1];
24 05  /* Befehlsworte */
25 iN  unsigned char bef[KOM][10]={
26 gN3    "dir",
27 Sv     "cd",
28 yV     "del",
29 7b     "mdir",
30 0b     "help",
31 zj     "cls",
32 5P     "view",
33 B4     "type",
34 5f     "save",
35 SK     "new",
```

```
Vq        "pfad",
xk        "ren",
6u        "endcli",
n9        "exit"
xt0  };
qi   /* Funktionsnamen der Befehle */
1K   int (*funk[KOM])() = {
fa3     &dir,
Vy      &cd,
xi      &del,
Gm      &mdir,
9m      &help,
yw      &cls,
Ea      &view,
KF      &type,
Eq      &save,
RX      &new,
e1      &pfad,
wx      &ren,
OJ      &ende,
dk      &ende
EAO  };
En   struct sgttyb stty;
aU   struct FileLock *verz;
Ob   struct FileInfoBlock *eintrag;
1b   /*          __________                   */
z6   /*              Funktionen               */
Ph   /*                            */
Zo   ren()
sB   /* Unbennen eines Eintrags - Dos-Funktion Rename() */
OT   {
903     int i=0;
GR      int j=0;
NU      int k=0;
3B      char c;
V1      if(parameter[0]=='\0'){
Cv6        printf("Aufruf: ren ALTER_NAME NEUER_NAME\n");
C1         return;
Ch3     }
ib      while(((c=parameter[i])!=0) && (j<2)){
LZ6        if(c==' '){
HC9           if(k!=0){
vJC              par[j][k]='\0';
Bz               if(strcmp(&par[j],"to")!=0) j++;
Kh               k=0;
Jo9           }
64            i++;
St            continue;
Mr6        }
Am         par[j][k]=c;
EE         k++;
B9         i++;
Qv3     }
6u      par[j][k]='\0';
Fx      if(j<1){
VE6        printf("Aufruf: ren ALTER_NAME NEUER_NAME\n");
V4         return;
V03     }
gP      if(!Rename(par[0],par[1])) printf("Kann nicht %s als %s
        umbenennen!\n",par[0],par[1]);
F1      else printf("%s heißt jetzt %s!\n",par[0],par[1]);
Y30  }
HT   pfad()
Wz   {
5n3     if(aktuellDir[0]=='\0') printf("Erst mit 'cd NAME' wird
        ein aktueller Pfad angelegt!");
kx      printf("%s\n",aktuellDir);
d80  }
dx   new()
05   /* löschen der Tastaturbelegung */
c5   {
Bh3     int i;
Px      for(i=0;i<95;i++) esc[i][0]='\0';
FG      printf("Tastenbelegung gelöscht!\n");
kFO  }
Nb   raw_on()
oe   /* ExCLI Übernimmt die Kontrolle Über die Tastatur */
jC   {
YF3     ioctl(0,TIOCGETP,&stty);   /* console-Parameter holen */
E3      stty.sg_flags |= RAW;      /* Parameter erweitern    */
rx      ioctl(0,TIOCSETP,&stty);   /* console-Parameter setzen *
        /
rMO  }
t7   raw_off()
3b   /* Amiga kontrolliert die Tastatur */
qJ   {
Ng3     ioctl(0,TIOCGETP,&stty);
iF      stty.sg_flags ^= RAW;
Bg      ioctl(0,TIOCSETP,&stty);
yTO  }
OD   ende()
Da   /* Programmende */
xQ   {
3Z3     printf("\n\n");
Xu      flag=0;
EX      run("RAM:C/assign C: SYS:c");
mP      run("del RAM:C/assign");
tA      run("del RAM:C/run");
YP      run("del RAM:C");
1F      raw_off();
17      Close(tastatur);
df      if(maschine) Close(drucker);
c0      exit(0);
ChO  }
Dj   save()
XF   /* Tastaturbelegung speichern */
Be   {
kG3     int i;
rC      if(parameter[0]!='\0')
JJ6        if(parameter[strlen(parameter)-1]!=':') strcat(parame
           ter,"/");
Lr3     strcat(parameter,"ExCLI.key");
xM      if(!(file=Open(parameter,MODE_NEWFILE))){ /* File öffnen
         */
Mb6        printf("Kann Tasten nicht speichern\n");
Nw         return;
Ns3     }
3f      for(i=0;i<95;i++) Write(file,esc[i],80);
RB      Close(file);
QvO  }
F4   type()
15   /* ähnlich dem CLI-Komando 'type' */
Ps   {
xJ3     unsigned char c;
5Y      char Dblock[512];
ap      int i=0;
8z6        int blocklaenge, zaehler;
uA3     if(parameter[0]=='\0'){
Vd6        printf("Aufruf: type FILE\n");
bA         return;
b63     }
Nx      if(!(file=Open(parameter,MODE_OLDFILE))){
Sx6        printf("Kann File nicht öffnen!\n");
fE         return;
fA3     }
Gt      cls();
3t8         do {
Aj6        blocklaenge = Read(file, Dblock, 512);/* Block  lesen
             */
fJ         for (zaehler = 0; zaehler < blocklaenge; zaehler++)
           {
Yx9           if(Dblock[zaehler] == '\n'){
NEC              putchar(Dblock[zaehler]);/* RETURN erlaubt */
YW               i++;
WU               if(i>19){
m7E                i=0;
r9                 printf("\n\033[2mTASTE - weiter | ESC - stop\
                   033[0m\n");
dI                 raw_on();
X7                 if((c=getchar())==27){
UdF                 blocklaenge=0;
WO                  raw_off();
py                  break;
vQE                }
L1                 else cls();
a4                 raw_off();
yTC              }
zU9           }
Pq            else if(Dblock[zaehler] >= 0x20)      /* < 32 ve
              rboten */
roC              putchar(Dblock[zaehler]);
2X6        }
OW3     } while (blocklaenge == 512);/* ein ganzer Block wurde g
        elesen */
vg      Close(file); /* File schliessen */
8H      putchar('\n');
6bO  }
```

Listing 1. »ExCLI« befreit Sie von den Widrigkeiten des CLI

```
view()
/* Tastaturbelegung einer einzelnen Taste anzeigen */
{
   int nr;
   if(parameter[0]=='\0'){
        printf("Aufruf: view TASTE\n");
        return;
   }
   if(parameter[0]>=32 && parameter[0]<=126){
      nr=parameter[0]-32;
      if(esc[nr][0]!='\0') printf("%s\n",esc[nr]);
      else printf("Nicht belegt!\n");
   }
   else printf("Taste nicht frei definierbar!\n");
}
mdir()
/* makedir-Kommando - Dos-Funktion CreatDir() */
{
   if(parameter[0]=='\0'){
      printf("Aufruf: mdir VERZEICHNISNAME\n");
      return;
   }
   if((verz=CreateDir(parameter))==0){ /* CreatDir - Dir an
legen */
     if(flag)
      printf("Verzeichnis '%s' konnte nicht angelegt werde
      n!\n",parameter);
   }
   else{
      UnLock(verz); /* freigeben der Lockstructur */
      if(flag) printf("Verzeichnis '%s' angelegt!\n",parame
      ter);
   }
}
del()
/* File löschen - Dos-Funktion DeleteFile() */
{
   if(parameter[0]=='\0'){
      printf("Aufruf: del FILENAME\n");
      return;
   }
   if(DeleteFile(parameter)==0){
      if(flag)
       printf("Eintrag %s konnte nicht gelöscht werden!\n"
       ,parameter);
   }
   else if(flag) printf("Eintrag %s gelöscht!\n",parameter)
   ;
}
dir()
/* Einträge eines Verzeichnisses listen */
{
   int i;
   /* Speicher fÜr FileInfoBlock schaffen */
   eintrag = (struct FileInfoBlock *) AllocMem(sizeof(struc
   t FileInfoBlock),0);
   if(parameter[0]=='\0') strcpy(store,aktuellDir);
   else strcpy(store,parameter);
      if((verz=Lock(store, ACCESS_READ))==0){
      printf("Finde Directory %s nicht!\n",store);
      return; /* nicht gefunden */
   }
   if(Examine(verz,eintrag)!=0){
      if(eintrag->fib_DirEntryType<=0){/* File oder Verze
      ichnis ? */
         printf("%s ist kein Verzeichnis!\n",store);
         return;
      }
   }
   else printf("Kann %s nicht listen!",store);
   printf("\n\033[4mEinträge von %s:\033[0m\n\n",eintrag->
   fib_FileName);
   ExNext(verz,eintrag);
   while (IoErr() != ERROR_NO_MORE_ENTRIES){ /* Dir-Einträg
   e lesen */
      if (eintrag->fib_DirEntryType > 0){  /* Directoryna
      me */
         printf("\033[2m%s\n\033[0m",eintrag->fib_FileName
         );
      }
      else{       /* Filename */
         printf("%s \r\t\t\t\t(%ld)\n",eintrag->fib_FileNa
         me,eintrag->fib_Size);
      }
      ExNext(verz,eintrag); /* nächster Eintrag */
   }
   FreeMem(eintrag,sizeof(struct FileInfoBlock)); /* Speich
   er freigeben */
   UnLock(verz);
}
cd()
/* festlegen einer neuen aktuellen Directory - Dos-Funktion
 CurrentDir() */
{
   int i;
   int bad=0;
   strcpy(store,aktuellDir); /* 'Sicherheitskopie' */
   eintrag=(struct FileInfoBlock *)AllocMem(sizeof(struct F
   ileInfoBlock),0);
   if(parameter[0]=='\0'){ /* nur cd */
       verz=Lock(aktuellDir, ACCESS_READ);
   Examine(verz,eintrag);
   printf("%s\n",eintrag->fib_FileName);
   }
   else{
   if (parameter[0]=='/') { /* aufsteigen mit 'cd /' */
     for (i=strlen(aktuellDir);aktuellDir[i]!='/'&&aktuellD
     ir[i]!=':';i--);
     if(aktuellDir[i] == ':') i++;
     aktuellDir[i] = '\0';
   }
   else { /* neue Directory */
     if((index(parameter,':'))==0){ /* kein ':' enthalten *
     /
        if(aktuellDir[strlen(aktuellDir)-1] != ':'){/* kein
        e Rootdir */
         strcat(aktuellDir,"/");
       }
       strcat(aktuellDir, parameter); /* neues Verzeichnis a
       nhängen */
     }
     else strcpy(aktuellDir, parameter); /* parameter enthä
     lt ':' */
   }
        if((verz = Lock(aktuellDir, ACCESS_READ)) != 0){
      if(Examine(verz,eintrag)!=0){
         if(eintrag->fib_DirEntryType<=0) bad++;
         else verz=CurrentDir(verz); /* neues Verzeichnis a
         nmelden */
      }
      else bad++;
        }
   else bad ++;
   if(bad) {   /* Das neue Verzeichnis existiert nicht! */
     printf("Finde Verzeichnis %s nicht!\n",parameter);
     strcpy(aktuellDir,store);/* RÜckkopieren der Sicherhe
     itskopie */
        }
   } /* else */
   UnLock(verz);
   FreeMem(eintrag,sizeof(struct FileInfoBlock)); /* Speich
   er freigeben */
}
_abort()
/* Prompt () schreiben. Der Namen _abort bewirkt, daß die
Tastenkombination
   CTRL-C nicht mehr zum Programmabbruch fÜhrt.
                          */
{
   p=-1;
   z=-1;
   printf("\033[33m \033[31m");
}
help()
/* Hilfsseite listen */
{
   int i;
   printf("\f\033[4mBefehle von ExCLI:\033[0m\n\n");
   printf("dir  - Verzeichnis listen        cd   - Arbeitsve
   rzeichnis festlegen\n");
   printf("del  - File löschen              mdir - Verzeichn
   is anlegen\n");
   printf("type - File als Text listen      help - Hilfsseit
   e listen\n");
   printf("cls  - Bilschirm löschen         view - Tastenbel
   egung prÜfen\n");
   printf("new  - Tastenbelegung löschen    pfad - komplette
   r Pfadname listen\n");
   printf("ren  - Eintrag umbenennen        exit - ExCLI bee
   nden\n");
```

```
Ho      printf("\n\033[4mFunktionstasten:\033[0m\n\n");
BY      for(i=0;i<9;i++) printf(" %d) %s\n",i+1,ftaste[i]);
RZA             printf("10) %s\n\n",ftaste[i]);
Pu0  }
1L   run(befehl)
ZC   /* Aufruf einer ExCLI-Funktion oder starten eines Programms */
w03     char *befehl;
Ps0  {
wP3     int i=-1;
MX      int j= 0;
ul      int eigen=0;
21      do{   /* Kommando und Parameter trennen */
KI6        i++;
gb         kommando[i]=befehl[i];
X93     }while((kommando[i]!=' ') && (kommando[i]!='\0'));
10      if(kommando[i]!='\0'){
nU6        kommando[i]='\0';
PN         i++;
Cn         while(befehl[i++]==' '); /* spaces Überlesen */
fh         i--;
BS         while((parameter[j++]=befehl[i++])!='\0');
hC3     }
c0      else parameter[0]='\0';
yt      /* Funktionstasten von links nach rechts shiften */
Sv      if(flag){
f66        for(i=0;i<9;i++) strcpy(ftaste[i],ftaste[i+1]);
uU         strcpy(ftaste[i],befehl);
nI3     }
dv      raw_off(); /* erlaubt Anhalten durch Space */
Ov      for (i=0;i<KOM;i++){
qc6        if(strcmp(kommando,bef[i])==0){
sU9           (*funk[i])();   /* Aufruf einer ExCLI-Funktion */
xG            eigen++;
t06        }
uP3     }
DQ      if(!eigen) Execute(befehl,0,0); /* Starten eines Programms */
Zo      raw_on(); /* Tastaturkontrolle an ExCLI */
1G      p=-1;
61      z=-1;
zU0  }
gz   checkkey(c)
Sf   /* checken der Funktionstasten und der Helptaste */
Vr3     unsigned char c;
zS0  {
7d3     int nr;
X0      int i=-1;
F8      unsigned char buff[50];
8m      /* Funktiontaste F1: \2330 */
1r      do{
vt6        i++;
TK         buff[i]=c;
qu         c=getchar();
7j3     }while((c!='~') && (c!='|'));
d1      if ((buff[i+1]=c)=='~' && i<1){
1D6        if (buff[0] == '?'){ /* Helptaste: \233? */
j69           flag=0;
ht            run("help");
nB            flag=1;
t4            _abort();
Kt            return;
Kp6        }
4S         if(buff[0]>='0' && buff[0]<='9'){ /* Funktionstasten */
1X9           nr=buff[0]-48;
vC            printf("\n\033[33m\033[31m %s",ftaste[nr]);
W3            strcpy(zeile,ftaste[nr]); /* in die Kommandozeile */
LS            z=strlen(zeile)-1;         /* kopieren */
02            p=z;
S1            return;
Sx6        }
Ty3     }
Uz0  }
10   c_del()
iq   /* Funktion wird von DEL und BACKSPACE benutzt */
Tw   {
2Y3     int i;
jx      for(i=p+1;i<=z;i++){
DC6        zeile[i]=zeile[i+1];
TY         printf("\b%c ",zeile[i]);
c73     }
gB      for(i-=p+1;i>0;i--) putchar('\b');
CV      z--;
fA0  }
0a   cls()
Yu   /* Bildschirm löschen */
e7   {
213     printf("\f");
kF0  }
Qe   insert()
qe   /* ein Space in die Kommanozeile einfÜgen */
jC   {
Io3     int i;
RJ      if((z==p) || (z+1>=MAXC)){ /* mehr als 80 Zeichen */
zQ6        putchar(7);        /* Blitz */
sR         return;
sN3     }
DS      z++;
zs      if(z>MAXC-6) putchar(7);   /* Blitz */
UU      cbef[0]=' ';          /* Space */
7t      for(i=p;i<=z;i++) cbef[i-p+1]=zeile[i+1]; /* Space einfÜgen */
AK      cbef[i-p+1]='\0';  /* cbef abschließen und Zeile neu schreiben */
bR      for(i=p;(zeile[i+1]=cbef[i-p])!='\0';i++) putchar(zeile[i+1]);
4x      for(i=z-p;i>0;i--) putchar('\b'); /* Cursor zurÜck */
0V0  }
i4   main()
yR   {
TH3     int i,nr;   /* Befehl-Nummer */
Xt      unsigned char c;
xA      tastatur=Open("RAW:",MODE_OLDFILE); /* Tastatur 'öffnen' */
CJ      run("mdir RAM:C"); /* Verzeichnis RAM:C anlegen */
BS      /* Ramdisk einrichten */
cR      printf("\n\n\033[3mRichte RAM-DISK ein ...\033[0m\n");
8z      run("copy SYS:C/run to RAM:C");
CR      run("copy SYS:C/assign to RAM:C");
oc      run("RAM:C/assign c: RAM:C");
TN      if(file=Open("ExCLI.key",MODE_OLDFILE)){ /* File zum Lesen öffnen */
9n6        printf("\033[3mLade KEY's ...\033[0m\n");
Gt         for(i=0;i<95;i++) Read(file,esc[i],80);
H1         Close(file);
G13     }
9E      printf("\233A\233A ______________________\n \033[2m\033[41m");
s4      printf(" -*- Gölzy's ExCLI -*- \033[40m\033[0m");
aA      printf("\n \n");
Po      run ("cd SYS:");
PZ      printf(" current dir:  SYS:\n");
w0      flag++;
Gz      raw_on(); /* Tastaturkontrolle Übernimmt ExCLI */
1e      _abort(); /* Prompt */
uv      for(;;) { /* Endlosschleife */
4K6        c=getchar(); /* Tastendruck */
rS         switch(c){
5b9           case 8:{ /* backspace */
iqC              if(p>=0){
irF                 p--;
SZ                  c_del();
W1C              }
Sb               break;
Y39           }
Pc            case '\t':{ /* tab */
soC              if(maschine){
R2F                 Close(drucker);
Jn                  maschine=0;
wz                  printf("\n\n \033[3m\033[4m'Schreibmaschine' OFF\033[0m\n\n");
FQ                  _abort();
fAC              }
JE               else{
TyF                 if(!(drucker=Open("PRT:",MODE_NEWFILE))){
05I                    printf("Finde keinen Drucker!\n");
KVF                 _abort();
kF                  }
0J                  else{
i2                  printf(" \033[3m\033[4m'Schreibmaschine' ON\033[0m\n\n");
a0                  maschine=1;
oJ                  }
pKC              }
lu               break;
```

Listing 1. (Fortsetzung)

```
        }
        case '\r':{/* wagenrÜcklauf=<RETURN> */
          printf("\n");
          z++;
          if(maschine){
            zeile[z]='\n';
            zeile[z+1]='\0';
            if(Write(drucker,zeile,strlen(zeile))<1)
              printf("\nDrucker einschalten!\n");
            p=-1;
            z=-1;
            break;
          }
          zeile[z]='\0';
          if(z>0) {
            run(zeile);
          }
          _abort();
          break;
        }
        case 27:{/* ESC */
          printf("\033[33m\233A\r\233K  'ESC' - Taste bel
          egen ");
          printf("oder 'Taste' - Kommando starten\033[31m
          \233B\r\233C\233C");
          c=getchar();
          if(c==27){
            printf("\033[33m\233A\r\233K  'Taste' - Komm
            ando speichern\033[31m\233B\r\233C\233C");
            c=getchar();
            if(c>=32 && c<=126){
              nr=c-32;
              if(z>=0){
                strcpy(esc[nr],zeile);
              printf("\nESC-'%c' gespeichert!\n",c);
              }
              else printf("Keine Kommandozeile?\n");
            }
            else printf("Taste kann nicht belegt werden!
            \n");
          }
          else{
            if(c>=32 && c<=126){
              nr=c-32;
              if(esc[nr][0]!='\0'){
                printf("%s\n",esc[nr]);
                run(esc[nr]);
              }
              else printf("Nicht belegt!\n");
            }
            else printf("Taste kann nicht belegt werden!
            \n");
          }
          _abort();
          break;
        }
        case 127:{ /* del-taste */
          if((z>=0) && (p<z)){
            printf("\233C");
            c_del();
          }
          break;
        }
        case 155:{/* CSI */
          c=getchar();
          if(c>='0' && c<='9' || c=='?'){
            checkkey(c);
          }
          else{
            if(c=='A') insert();/* C up */
            if((c=='B') && (z>=0) && (p<z)){/* C down
            */
              printf("\233C");
              c_del();
            }
            if(c=='D' && p>=0){/* C left */
              printf("\b");
              p--;
            }
            if(c=='C' && p<z){/* C right */
              printf("\233C");
              p++;
            }
            if(c=='S'){/* C shift down */
              if(p>=MAXC-6)
                printf("\r\233K\233A\233C\233C\233K");
              else printf("\233B\r\233K\233A\233C\233C\
              233K");
              p=-1;
              z=-1;
              zeile[0]='\0';
            }
            if(c=='T'){/* C shift up */
              if(p>=MAXC-6)
                printf("\233K\233A\r\233C\233C");
              else printf("\233K\233B\r\233K\233A\233C\
              233C");
              z=p;
              p=-1;
              zeile[z+1]='\0';
              for(;p<z;p++) printf("\233C");
            }
            if(c==' '){
              c=getchar();
              if(c=='A'){/* C shift left */
              for(;p>=0;p--) printf("\b");
              }
              if(c=='@'){/* C shift right */
              for(;p<z;p++) printf("\233C");
              }
            }
          }
          break;
        }
        case 255:{/* CTRL-d und Neuaufruf */
          printf("exit");
          ende();
          }
        default:{/* Kommandozeile aufbauen */
          /* z=Länge der Zeile; p=Cursorposition darin */
          if(z==p) z++;
          p++;
          if((p==z) && (z>MAXC-6)) putchar(7);
          if(z<MAXC){
            zeile[p]  = c;
            zeile[z+1]= '\0';
            putchar(c);
          }
          else{
            if(p==z) z--;
            p--;
          }
          break;
        }
      } /* switch */
    } /* while */
}
(C) 1988 M&T
```

Listing 2. (Schluß)

Programmname:	ExCLI.bat
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec
Aufrufe:	

```
Programmautor: Arno Gölzer
--------------------------

; -*- Gölzy's ExCLI BATCHFILE -*-
;
  makedir RAM:C
  copy SYS:C/copy RAM:C
  copy SYS:C/info RAM:C
  copy SYS:C/list RAM:C
  copy SYS:C/execute RAM:C
; usw ...
; ExCLI aufrufen
  path SYS:C
  ExCLI
(C) 1988 M&T
```

Listing 2. »ExCLI.bat« ist ein Beispiel für eine Batchdatei zum Aufruf dieses Tools

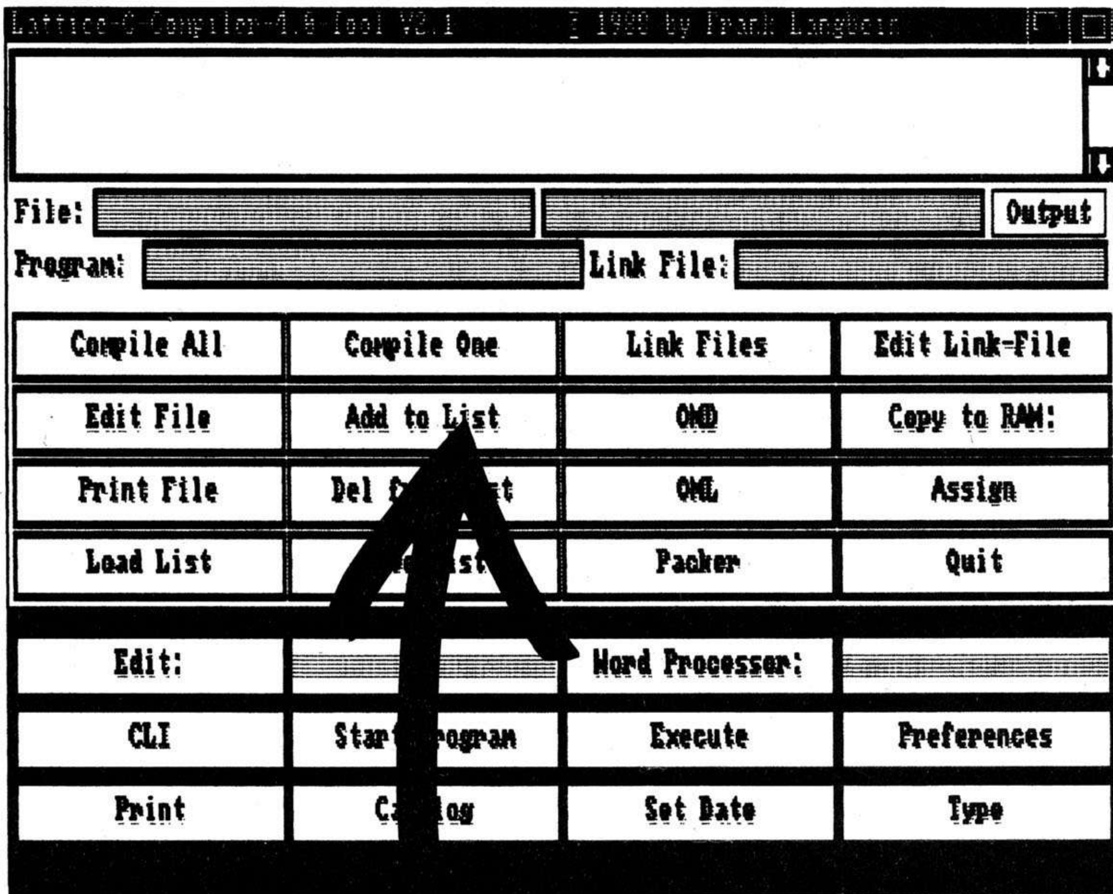


Bild 1. Mit »LCT« bedienen Sie Ihren Compiler ausschließlich über Gadgets

Compilieren à la carte

Der Lattice-C-Compiler galt lange als gutes Beispiel für schlechten Ton – zumindest in der Benutzerfreundlichkeit. Mit unserem Utility gehören diese Gerüchte endgültig der Vergangenheit an.

Am Anfang war ... nein, nicht was Sie denken! Gemeint ist der Anfang des Amiga. Und da war Lattice. Ein toller C-Compiler – aber mit anstrengender Bedienung. Die ellenlangen Funktionsaufrufe haben so manchem Programmierer ein Stück Nervenkraft abgefordert. Dieser Streß hat ein Ende – jetzt gibt es »LCT«.

Mit LCT wird die Bedienung des Lattice-C-Compilers radikal vereinfacht. War früher ein Compiler-Aufruf oder die Nutzung der vielfältigen Zusatzfunktionen den Fingerübungen eines Pianisten gleichzusetzen, so genügen jetzt wenige Mausklicks. Und durch die vorbildliche Kommentierung des Quellcodes sind Anpassungen an die eigenen Bedürfnisse leicht durchführbar.

Das Listing von LCT hätte gedruckt 13 Seiten in Anspruch genommen. Aufgrund der Länge haben wir uns entschlossen, LCT nur auf Diskette anzubieten. Sie erhalten die Programm-Service-Diskette mit dem Bestellschein auf Seite 34.

Kopieren Sie das File »LCT« in das C-Directory Ihrer Lattice-Systemdiskette und Ihre Leiden haben ein Ende. Mit

```
LCT [Editor [WProzessor
[Programm [CLI]]]]
```



starten Sie Ihre neue Benutzeroberfläche. Sollte sich im Namen eines der Parameter ein Leerzeichen befinden, so müssen Sie diesen in Anführungszeichen übergeben. Alle Parameter sind optional. Sie müssen – wenn sie angegeben sind – in der oben genannten Reihenfolge stehen. Weil das Leerzeichen als Trennzeichen zwischen den Parametern dient, ist es zwar möglich, nur die Parameter für Editor und WProzessor (siehe unten) zu übergeben; WProzessor und CLI würden dagegen nicht erkannt. »LCT ?« gibt Ihnen jederzeit die korrekte Syntax aus.

Die Parameter von LCT ermöglichen Ihnen das Integrieren beliebiger Programme in diese Benutzeroberfläche. Dabei sollten Sie sich an folgende Konventionen halten:

Parameter sorgen für Flexibilität

Editor ist der Name des Editors, auf den die Befehle »Edit File«, »Edit Link-File« und »Edit« zugreifen sollen. Geeignet ist jeder Editor, der beim Aufruf den Namen eines zu bearbeitenden Files akzeptiert. Wird kein Parameter angegeben, verwendet LCT den »Ed« aus dem C-Directory.

WProzessor gibt ein Textverarbeitungsprogramm an, das von dem Gadget »Word Processor« aktiviert werden soll. Auch hier kann jedes Programm verwendet werden, das die Übergabe des zu bearbeitenden Files als Argument erlaubt. Voreingestellt ist »Notepad«.

Programm ist ein beliebiges Programm, das von LCT per Gadget aufgerufen werden soll. In diesem Gadget wird auch der übergebene Programmname eingetragen. Fehlt dieser Parameter beim Aufruf, so wird »Preferences« eingetragen.

CLI steht für den Namen des Befehls, mit dem LCT einen neuen Task startet. Es kann jedes Programm aufgerufen werden, das keine Parameter benötigt, oder immer die gleichen – die beim Aufruf von LCT mit dem Namen übergebenen. Im Gadget steht allerdings immer »CLI«. Wird hier kein Parameter angegeben, ruft das CLI-Gadget den Befehl »NEWCLI CON: 0/176/640/90/CLI« auf.

Ein Aufruf von LCT könnte beispielsweise so aussehen:

```
LCT ed Beckertext df2:
Kermit "NEWCLI NEWCON:
10/10/100/100/SHELL"
```

oder

```
LCT df1:SuperEd dh0:
Utilities/MyTexter
```

Die voreingestellten Parameter sind in »MyIncludes.h« definiert. Der Name für den Editor steht in der Konstanten »Editor [101]« und für die Textverarbeitung in »WProcessor[101]«. »Pref[101]« gibt den Namen des Programms an, und CLI ist in »CLIName[101]« definiert.

Sie können LCT eine eigene »Assign-Liste« abarbeiten lassen. Diese wird automatisch im File »LCT-ASSIGN« gesucht. Ist dieses File nicht im aktuellen Directory, so werden alle Assign-Definitionen aus den zuletzt eingestellten Werten (beispielsweise von der Startup-Sequence) übernommen.

Eine Startup-Sequence für den LCT finden Sie in Bild 2. Diese verwendet die gleichen Definitionen, die bei LCT voreingestellt sind. Beachten Sie bitte, daß diese Voreinstellungen nicht automatisch durchgeführt werden, sondern beim Anklicken des »Assign«-Gadgets. Die Voreinstellungen für den Assign-Befehl lauten:

```
INCLUDE: Lattice
_C_4.0.2:CompactH
LC: C:
LIB: Lattice
_C_4.0.2:lib:
QUAD: RAM:
```

Diese Definitionen können Sie in »MyIncludes.h« verändern. Sie stehen im Feld »As[4][101]« in der genannten Reihenfolge. Falls Sie das Programm an einen anderen Compiler anpassen und andere Devices definieren wollen (beispielsweise für die Version 5.0): Die Namen finden Sie im Feld »Ev[4][9]«.

In Bild 1 sehen Sie den Bildschirmaufbau von LCT. Oben befindet sich ein Feld, das zu Beginn leer ist. Erst durch das Laden einer File-Liste oder Eingabe von File- und Programmnamen in den entsprechenden Gadgets und Anklicken von »Add to List« füllt sich der Kasten. Links sind die Filenamen und rechts die zugehörigen Programmnamen zu sehen.

Mit Hilfe dieser Liste haben Sie immer einen Überblick über die Files, aus denen sich Ihr Programm zusammensetzt. Sie können sogar mehrere Programmprojekte gleichzeitig verwalten – es sind 50 Einträge erlaubt. Mit den beiden Pfeilen wird die Liste gescrollt.

Wenn Sie mit der Maus ein File anklicken, erscheint es im Gadget »File:« und der entsprechende Programmname im Gadget »Program:«.

Neben dem Eingabefeld »File« werden die Compiler- und Assembler-Anweisungen angezeigt, die für dieses File definiert wurden. Änderungen der Anweisungen, die Sie in diesem Gadget vornehmen, werden in die Programmliste übernommen; Änderungen im File- oder Programm-Gadget (Mauszeiger ins Gadget – anklicken – Eintrag ändern – <RETURN> drücken) werden erst dann aktualisiert, wenn Sie das Feld »Add to List« anklicken.

C-Source-Files müssen mit ».c« enden. Bei jeder anderen Endung vermutet LCT ein Assembler-File.

Im Gadget »Link File:« tragen Sie den Namen jenes Files ein, in dem sich die Parameter für den BLink befinden. Der Name dieses Files ist unabhängig vom Programmnamen. Wenn das angegebene Link-File noch nicht vorhanden ist, wird der Editor aufgerufen und Sie können das benötigte File zusammenstellen.

Menü aus Gadgets

LCT lenkt alle Ausgaben des Lattice in ein File namens »LCT.output« um. Nach Anklicken des Gadgets »Output« wird dieses File angezeigt. Ist der Inhalt des Files größer als eine Bildschirmseite, so stoppt die Ausgabe. Mit der linken Maustaste wird die Ausgabe fortgesetzt, mit der rechten abgebrochen. »LCT.output« befindet sich immer im aktuellen Directory und wird von der »Output«-Funktion auch nur dort gesucht. Wurde das aktuelle Directory gewechselt, müssen Sie statt »Output« das Gadget »Type« verwenden.

Die Compilerfunktionen und die File-Liste werden über den Gadget-Block im mittleren Bildschirmbereich gesteuert:

Compile All compiliert beziehungsweise assembliert alle Files, deren Programmname der Anzeige in »Program« entspricht. Die Reihenfolge wird durch die Fileliste bestimmt. Ob ein File ein C-Source- oder ein Assemblerfile ist, erkennt LCT an der Endung (».c« steht für C, alles andere für Assembler). Auf Wunsch werden die Files vor dem Compilieren ins RAM kopiert und nur das Ergebnis-File auf Diskette geschrieben. Über ein String-Gadget am unteren Rand werden die Steueroptionen eingegeben, die für alle Files verwendet werden.

Während des Compilierens wird das LCT-Fenster geschlossen und die Anzahl der Bitplanes auf Eins gesetzt – das spart RAM und Zeit.

Compile One richtet sich nur nach dem Inhalt von »File:« und »Program:«. Ob Compiler oder Assembler verwendet werden, entscheidet sich durch die File-Endung. Bevor das Compilieren beginnt, werden Sie auch hier gefragt, ob das File ins RAM kopiert werden soll. Das compilierte File wird dann wieder auf die Diskette kopiert.

Link Files ruft den Linker »BLink« auf. Das »WITH«-File wird aus dem entsprechenden Gadget eingelesen, die Meldungen des Linkers gehen nach »LCT.output«. Wenn BLink unbekannte Funktionen findet, wird dies zwar in »LCT.output« vermerkt, aber die Eingabe für Ersatznamen entfällt, da BLink mit der Option »BATCH« aufgerufen wird.

Edit Link-File und **Edit File** helfen beim bequemen Editieren der in den jeweiligen Eingabefenstern stehenden Files.

Print File druckt das in »File:« eingetragene Programmfile wahlweise als ASCII- oder Hex-Dump und mit oder ohne Zeilennumerierung. LCT verwendet dazu den CLI-Befehl »Type«, der deshalb im C-Directory vorhanden sein muß.

Add to List fügt den Inhalt von »File:« und »Program:« an die File-Liste an. Wenn das Programmfile zwar vorher aus der Liste ausgewählt, aber der Name geändert wurde, betrachtet LCT dies als neues File. Wird in »Program:« keine Bezeichnung eingetragen, so wird »NoName« in die Liste und in dieses Gadget eingesetzt.

Del from List löscht das Programmfile, das zuvor angeklickt wurde, aus der Liste. Anklicken ist dabei nötig – die direkte Eingabe in das Feld genügt nicht.

OMD aktiviert den Lattice-Object-Module-Disassembler für das in »File:« stehende File. Das Ergebnis finden Sie als »Filename.lst« im aktuellen Directory. Das zugehörige Objektfile »Filename.o« muß sich dabei auf der Diskette im aktuellen Directory befinden, da es von OMD verwendet wird. Die OMD-Optionen werden über ein Eingabefeld festgelegt, das am unteren Rand eingeblendet wird. »OMD« ist ein typisches Beispiel dafür, wie LCT die Bedienung des Lattice vereinfacht. Ein kurzes »KLICK-KLICK« ersetzt den Aufruf »OMD [listfile] [options] objfile [textfile]«.

```
CHO ; StartUp-Sequence for Lattice-C-Compiler and LCT V2.1
NI  ; (c) 1988 by Frank Langbein
yC  AddBuffers df0: 30      ; Buffer für Laufwerk df0: erhöhen
xa  path df0:system add     ; Pfad hinzufügen
T7  ; Datum umd Zeit abfragen
CO  settime
Ex  date >S:System-Date
ip  ; ASSIGN-Definitionen setzen
Iu  copy C:Assign RAM:
yX  ram:assign QUAD:    RAM:
sM  ram:assign LC:      C:
5H  ram:assign INCLUDE: Lattice_C_4.0.2:CompactH
9Q  ram:assign LIB:     Lattice_C_4.0.2:lib
7Z  delete Ram:Assign
uV  Lattice_C_4.0.1:PopCLI 300      ; PopCLI starten
f0  run conman >NIL: -q             ; Conman starten
mk  assign LCT: SYS:LCT             ; LCT: zum Starten von LCT def
    inieren
Ba  stack 15000                     ; Stack erhöhen
RP  run LCT:                        ; LCT als weiteren Task starte
    n
ID  quit                            ; Zum CLI
(C) 1989 M&T
```

Bild 2. Ein Beispiel für eine Startup-Sequence, die alle »Assigns« einstellt und LCT startet

OML startet das gleichnamige Lattice-Programm. Das Resultat finden Sie in »LCT.output«, das Sie sich mit einem Klick auf »Output« auf den Bildschirm holen. Die Optionen werden aus dem Feld für die Programmoptionen (rechts oben neben »File:«) übernommen. Der Name der Bibliothek muß im Feld »Program:« stehen. Über das String-Gadget am unteren Rand geben Sie die Befehle zur Steuerung von OML ein. **Vorsicht!** Sie sollten hierbei nie die Eingabe der Befehle auf »stdin« legen (mit der »@«-Option), LCT reagiert hier sehr empfindlich und holt den Guru zu Hilfe.

Packer ruft »ICompact« auf. Als Eingabefile wird das in »Program:« stehende File verwendet. Den Namen des Zielfiles müssen Sie über das Gadget am unteren Rand des Bildschirms eingeben, das nach dem Anklicken von »Packer« eingeblendet wird. Das kompaktierte File kann nicht mehr editiert, sondern nur noch vom Lattice-C-Compiler gelesen werden.

Copy to RAM: kopiert das File im Feld »File:« ins RAM und setzt in der File-Liste den Zusatz »RAM:«. Es findet jedoch kein Überschreiben statt. Wenn ein File ins RAM kopiert werden soll, dessen Name dort bereits existiert, wird der Aufruf zurückgewiesen. Sie erhalten dabei keine Fehlermeldung.

Assign erlaubt Ihnen die Eingabe der Parameter für den »Assign«-Befehl des CLI über ein Eingabefenster. Natürlich muß dieser Befehl im C-Ordner liegen. Die Voreinstellungen wurden bereits zu Beginn dieses Artikels beschrieben.

Ihre Eingaben werden in dem File »LCT-Assign« im aktuellen Directory gespeichert. Sie stehen dort in der angegebenen Reihenfolge und sind jeweils 101 Zeichen lang. »LCT-Assign« wird von LCT angelegt, wenn es noch nicht vorhanden war, ansonsten überschrieben. Fehleingaben werden zwar gespeichert, aber nicht ausgeführt – die alte Einstellung bleibt dann erhalten.

Save List speichert die aktuelle File-Liste unter einem einzugebenden Namen. Es werden die komplette File-Liste und der Inhalt von »Link-File:« gespeichert.

Load List ist das Gegenstück zu »Save List«.

Im unteren Teil des Bildschirms finden Sie weitere Hilfsfunktionen:

Edit: erlaubt, ein beliebiges File mit dem voreingestellten Editor zu bearbeiten. Den Namen des zu editierenden Files geben Sie im Feld neben dem Gadget ein. Ist das Feld leer und Sie klicken »Edit:« an, geschieht nichts. Sobald Sie einen Namen in dieses Feld schreiben und <RETURN> drücken, wird der Editor aufgerufen. Steht bereits ein Name in dem Feld, so genügt ein Anklicken von »Edit:«, um den Editor zu starten.

Word Processor arbeitet wie »Edit:«, allerdings wird hier die Textverarbeitung aufgerufen (wenn Sie die Aufrufparameter nutzen, bleibt es natürlich Ihnen überlassen, welches Programm hier gestartet wird).

CLI ruft einen neuen CLI-Task auf, der unabhängig von LCT läuft und auch nach Beenden dieses Programms erhalten bleibt.

Start Program startet ein beliebiges Programm, dessen Namen Sie über ein Eingabefenster am unteren Rand des Bildschirms eingeben. Auch Parameter, die Sie hier angeben, werden an das aufzurufende Programm übergeben.

Execute arbeitet wie »Start Program«, allerdings mit Batch-Dateien.

Preferences steht als Beispiel für ein beliebiges, beim Aufruf an LCT übergebenes, Programm, dessen Name dann in diesem Gadget dargestellt wird. Voreingestellt ist das Programm »Preferences«. Wenn Sie mit Preferences die Farben des Bildschirms ändern wollen, beachten Sie bitte, daß dies keinen Einfluß auf LCT hat. LCT definiert seine eigene Farbtabelle, die Sie in »MyIncludes« finden (»colormap[8]«) und nur dort ändern können.

Print benutzt den CLI-Befehl »Type« und gibt ein beliebiges File auf den Drucker aus. Die Eingabe des Namens erfolgt über ein Eingabefeld am unteren Rand des Bildschirms, das nach dem Anklicken von Print eingeblendet wird. Über zwei Requester werden Sie gefragt, ob ein Hex-Dump ausgegeben werden soll und ob eine Zeilennumerierung gewünscht ist.

Catalog zeigt den Inhalt eines frei wählbaren Directory an. Wird keines angegeben, sondern das String-Gadget nur mit <RETURN> bestätigt, so wird das aktuelle Directory gelesen. Existiert das gewählte Directory, so wird ein zusätzliches Fenster geöffnet und die Files mit den gesetzten »Protection«-Flags und der Filegröße in Bytes angezeigt. Sollten Sie ein Directory anfordern, das nicht existiert, dürfen Sie zur Belohnung eine Fehlermeldung quittieren. Sind mehr Einträge vorhanden, als auf einer Bildschirmseite dargestellt werden können, so stoppt die Ausgabe und kann mit einer Maustaste fortgesetzt werden.

Set Date benutzt den CLI-Befehl »Date« (der im C-Ordner liegen sollte) und verlangt die bekannte Eingabesyntax.

Type gibt ein File auf den Bildschirm aus, dessen Name (und gegebenenfalls Pfad) Sie im String-Gadget eingeben.

Auch ein Super-Tool wie LCT hat kleine Schwächen:
- LCT verändert die Anzahl der »BitPlanes« des Workbench-Screens. Wenn Sie LCT mehrmals aufrufen, kann es sein, daß die Anzahl der BitPlanes und damit die Farben nicht mehr stimmen. Lassen Sie sich davon aber nicht beirren – die Funktionen von LCT sind davon nicht beeinträchtigt.
- Ist LCT mit einem Befehl beschäftigt, so wird dies durch einen Mauszeiger angezeigt, der auf einen Pixel reduziert ist. Klicken Sie Gadgets nur an, wenn der Mauszeiger die gewohnte Form hat. Ansonsten kann es zu unvorhersehbaren Ergebnissen kommen.
- CLI-Fehlermeldungen (beispielsweise durch ein beim Aufruf falsch definiertes Fenster verursacht) werden von LCT nicht angezeigt – es passiert einfach gar nichts.
- Wenn Sie eine Eingabe über ein String-Gadget machen, muß diese mit <RETURN> quittiert werden. LCT erkennt sonst die Eingabe nicht.
- Bei LCT gibt es kein Zurück! Wenn Sie eine Funktion aktivieren, wird diese erbarmungslos zu Ende geführt – es gibt keine Abbruchmöglichkeit.

Sicher kennen Sie die Shortcuts der Workbench-Requester. LCT erlaubt Ihnen den Gebrauch dieser Hilfen in den String-Gadgets: (siehe Kasten auf dieser Seite).

Entsteht ein Fehler bei der Ausführung eines Befehls, so wird am oberen Bildschirmrand ein Requester geöffnet und die Fehlermeldung oder Fehlernummer ausgegeben. Durch Anklicken des »OK«-Gadgets quittieren Sie und löschen damit den Requester.

Mit dem LCT steht Ihnen ein Werkzeug für den Lattice-Compiler zur Verfügung, das Sie sicher schon bald nicht mehr missen wollen – LCT macht aus dem C-Compiler einen »KLICKompiler«. LCT ist fast unbegrenzt flexibel und erweiterungsfähig. Vieles läßt sich über die Aufrufparameter realisieren. Wem das nicht reicht, der kann LCT über die Definitionen im Include-File verändern.

(Hans Waldhäusel/so)

<CRSR-links/rechts>	Cursor im String bewegen
<SHIFT CRSR-li/re>	Cursor zum String-Anfang/-Ende
<DEL>	Zeichen unter dem Cursor löschen
<BACKSPACE>	Zeichen links vom Cursor löschen
<AMIGA-rechts X>	Zeile löschen
<AMIGA-rechts Q>	letzte Veränderung rückgängig machen
<RETURN>	Eingabe beenden

Die Maus als

Aztec-C-Programmierer mußten bislang ohne den Komfort der Workbench auskommen. Wer keine Shell hat, muß bei jedem Aufruf die Kommandozeile neu eingeben. Schluß damit: Mit »Cont« bedienen Sie Ihren Compiler über die Maus.

Wer ärgert sich nicht über die Fingerakrobatik, die besonders dem C-Programmierer vom CLI auferlegt wird. Ellenlange Optionslisten, die bei jedem Compiliervorgang oder Tippfehler neu eingegeben werden müssen, gehen dem Programmierer gewaltig auf die Nerven. Von jetzt an können Sie sich zurücklehnen und entspannen: Zum Compilieren reicht es, wenn Sie eine Hand an der Maus haben. Vergessen Sie den Ärger über falsch eingegebene Kommandozeilen, über die »Updaterei« Ihres Makefiles, über das »Durchscrollen« und Ändern der letzten 20 Befehle in der Shell.

»Cont« (Listing 2) öffnet ein eigenes Fenster als »Schaltpult« (Bild 1). Es besteht aus insgesamt 30 Gadgets (6 String- und 24 Boolean-Gadgets), die anfangs teilweise nicht wählbar sind, was Sie an der Geisterschrift erkennen können. Erst wenn Sie ein File als »Mainfile« angegeben haben, macht es Sinn, dessen Quellcode zu editieren, auszudrucken oder zu compilieren.

Geben Sie Listing 1 (»Contwind.h«) und Listing 2 ein, compilieren Sie »Cont« mit

```
cc Cont
```

und linken Sie mit

```
ln Cont -lc
```

»ContWind.h« wird dabei automatisch eingebunden. Es muß im aktuellen Verzeichnis liegen. Um die »ClearScr«-Funktion nutzen zu können, benötigen Sie ein solches Programm, welches das CLI-Fenster löscht und das Prompt in die linke obere Ecke setzt. Falls Sie ein solches Utility besitzen, kopieren Sie es in Ihren C-Ordner und nennen es »CLS«. Wenn nicht, tippen Sie Listing 3 ab und übersetzen Sie es mit

```
cc CLS
ln CLS -lc
```

Daneben greift Cont auf die Befehle »type«, »dir« und »Ed« zu. Wenn Sie den Amiga-DOS-Ed nicht verwenden wollen, kopieren Sie einfach einen anderen Editor in den C-Ordner und nennen ihn »Ed«. Kopieren Sie Cont in den C-Ordner ihrer Startdiskette, legen Sie ein Directory »:ContDir« an und ergänzen Sie die »Startup-sequence« um die Zeile

```
run cont
```

Booten Sie jetzt von dieser Diskette, dann sehen Sie das oben beschriebene »Aztec-C Control-Window V4.0« auf dem Bildschirm. Tragen Sie hinter »Mainfile« Ihr Programm ein und die Arbeit kann beginnen:

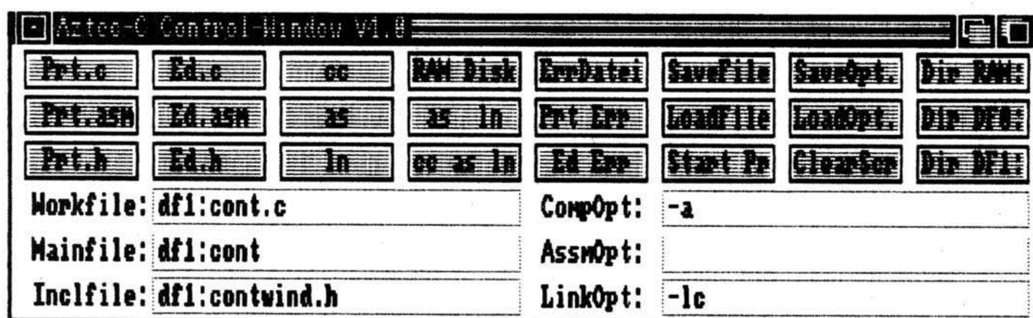


Bild 1. 30 Gadgets bilden das »Schaltpult« für Ihren Aztec-Compiler

Hinter »Workfile« wurde automatisch der Name Ihres Programms mit der Endung ».c« eingetragen. Wenn Sie Includes bearbeiten wollen, müssen Sie diese selbst angeben (hinter »InclFile«).

Starten Sie mit einem Klick auf das Gadget »Ed.c« den Editor mit Ihrem Sourcecode. Wollen Sie das eingestellte Includefile bearbeiten, klicken Sie einfach auf »Ed.h«. »Ed.asm« ist für das Nachbearbeiten des Assemblercodes gedacht. Cont ersetzt hier selbständig die Endung ».c« durch ».asm«. Wenn Sie mit dem Bearbeiten des Sourcecodes fertig sind, speichern Sie und verlassen den Editor. Cont wartet auf das Beenden des Editors, bevor mit dem Compilieren begonnen werden kann. Wer den Editor lieber ständig im Speicher hält, entfernt die Zeilen 143 und 145 (»SwitchGadgets«) aus dem Programm.

Starten Sie den Compiler mit einem Klick auf das »cc«-Gadget. Cont startet darauf den Aztec-Compiler mit den voreingestellten Optionen (hinter »CompOpt« können Sie diese ablesen). Wollen Sie andere Optionen verwenden, klicken Sie nur auf das String-Gadget und tragen Ihre Optionen hier ein. Dasselbe gilt für die Assembler- und Linker-Optionen in den Zeilen darunter. Hat der Compiler seine Arbeit beendet, so können Sie die Fehlermeldungen auslesen oder mit »as« den Assembler starten beziehungsweise danach mit »ln« den Linker.

Um sich die Arbeit zu vereinfachen, können Sie aber auch das »as ln«- oder »cc as ln«-Gadget verwenden. Mit letzterem sollten Sie allerdings etwas vorsichtig umgehen, denn Cont respektiert Ihre Wünsche kritiklos – es startet Assembler und Linker auch dann, wenn beim Compilieren noch Fehler aufgetreten sind.

Apropos Fehler: Sie können die Fehlermeldungen in eine Error-Datei umlenken, indem Sie das Gadget »ErrDatei« betätigen. Um diese Datei auszulesen, klicken Sie auf »Ed Err« oder »Prt Err«. Im ersten Fall werden die Fehlermeldungen in den Editor geladen, im zweiten über den Drucker ausgegeben.

Auf ähnliche Weise geben Sie Sourcefiles auf den Drucker aus: »Prt.c« druckt das Workfile aus, »Prt.asm« und »Prt.h« das Assemblerlisting beziehungsweise das Headerfile. Wenn Sie beim Compilieren die RAM-Disk nutzen wollen, klicken Sie einfach »RAM Disk« an. Darauf verfärbt sich dieses Gadget. Von jetzt an werden alle Ausgaben auf die RAM-Disk umgeleitet.

Am rechten Rand des Fensters sehen Sie die Gadgets »Dir RAM:«, »Dir DF0:« und »Dir DF1:«. Diese führen den eingetragenen Befehl aus und zeigen das Ergebnis im CLI-Fenster an. Schließen Sie nie das CLI nach dem Start von Cont, die Folge wäre eine »Privilegverletzung« mit der Guru-Nummer »00 00 0008«.

Natürlich müssen Sie die Optionen nicht bei jedem Cont-Start neu eintragen. Klicken Sie auf »SaveOpt.« und Ihre Optionen werden im File »cont.opt« im Directory »ContDir« gespeichert. Beim nächsten Start können Sie nach einem Klick auf »LoadOpt.« mit den alten Einstellungen weiterarbeiten. Das gleiche gilt für die Einstellung der Arbeitsdateien. Mit »Save-File« speichern Sie die Einstellungen und mit »LoadFile« sind sie beim nächsten Start wieder verfügbar.

Mit »Start Pr« starten Sie einen Testlauf des im Mainfile eingestellten Programms.

Mit der hier vorgestellten Oberfläche sollten Sie in der Lage sein, annähernd alle Programmiersituationen per Mausklick zu meistern. Wem die Funktionen dennoch nicht ausreichen, der sollte sich Listing 1 und 2 aufmerksam anschauen. Es dürfte nur geringe Probleme bereiten, das Fenster etwas zu vergrößern und eigene Funktionen einzubauen – sehen Sie sich dazu die Gadget-Definitionen in »ContWind.h« an sowie die Abfragen in Zeile 76 bis 288 von »Cont.c«. In jedem Fall haben Sie mit Cont ein Werkzeug, das Ihnen endlich auch beim Programmieren den ersehnten Workbench-Komfort bringt.

(Wilfried Siegbert/so)

Steuermann

```
/* äöü Drucker umschalten */
/*****************************************************************/
/*                     Includefile für Cont.c                    */
/*****************************************************************/
/*                                  *                            */
/*             Name: ContWind.h     *           H.J.P.           */
/*            Datum: Oct./Nov. 88   *                            */
/*****************************************************************/
struct Library *IntuitionBase;
struct Library *GfxBase;
/* Font Definition */
   struct TextAttr Font = {
      (UBYTE *)"topaz.font",
      TOPAZ_EIGHTY,
      FS_NORMAL,
      FPF_ROMFONT
   };
/* Border Definition */
   SHORT GadgVectors[] = { 0, 0, 69, 0, 69, 11, 0, 11, 0, 1 };
   struct Border GadgBorder = {
      -3, -2, GBFRONTPEN, GBBACKPEN, JAM1, 5,
      (SHORT *)GadgVectors,
      NULL
   };
/* Prt.c Gadget */
   struct IntuiText PrtCText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)" Prt.c  ",
      NULL,
   };
   struct Gadget PrtCGadget = {
      NULL,
      13, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &PrtCText,
      0, NULL, 0, NULL
   };
/* Ed.c Gadget */
   struct IntuiText EdCText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)" Ed.c   ",
      NULL,
   };
   struct Gadget EdCGadget = {
      &PrtCGadget,
      92, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &EdCText,
      0, NULL, 0, NULL
   };
 /* Cc Gadget */
   struct IntuiText CcText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)"   cc   ",
      NULL,
   };
   struct Gadget CcGadget = {
      &EdCGadget,
      171, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &CcText,
      0, NULL, 0, NULL
   };
/* RAM Disk Gadget */
   struct IntuiText RamDiskText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)"RAM Disk",
      NULL,
   };
   struct Gadget RamDiskGadget = {
      &CcGadget,
      250, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE | TOGGLESELECT,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &RamDiskText,
      0, NULL, 0, NULL
   };
/* SaveOpt Gadget */
   struct IntuiText SaveOptText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)"SaveOpt.",
      NULL,
   };
   struct Gadget SaveOptGadget = {
      &RamDiskGadget,
      487, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &SaveOptText,
      0, NULL, 0, NULL
   };
/* Save File Gadget */
   struct IntuiText SaveFileText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)"SaveFile",
      NULL,
   };
   struct Gadget SaveFileGadget = {
      &SaveOptGadget,
      408, 14, 64, 8,
      GADGHCOMP,
      RELVERIFY | GADGIMMEDIATE,
      BOOLGADGET,
      (APTR)&GadgBorder,
      NULL,
      &SaveFileText,
      0, NULL, 0, NULL
   };
/* ErrDat Gadget */
   struct IntuiText ErrDatText = {
      GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
      (UBYTE *)"ErrDatei",
      NULL,
   };
   struct Gadget ErrDatGadget = {
      &SaveFileGadget,
```

Listing 1. In »ContWind.h« werden Ihnen die Definitionen der Datenstrukturen abgenommen

```
        329, 14, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE | TOGGLESELECT,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &ErrDatText,
        0, NULL, 0, NULL
    };
/* DirRam Gadget */
    struct IntuiText DirRamText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"Dir RAM:",
        NULL,
    };
    struct Gadget DirRamGadget = {
        &ErrDatGadget,
        566, 14, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &DirRamText,
        0, NULL, 0, NULL
    };
/* Prt.asm Gadget */
    struct IntuiText PrtAsmText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)" Prt.asm",
        NULL,
    };
    struct Gadget PrtAsmGadget = {
        &DirRamGadget,
        13, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &PrtAsmText,
        0, NULL, 0, NULL
    };
/* Ed.asm Gadget */
    struct IntuiText EdAsmText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)" Ed.asm ",
        NULL,
    };
    struct Gadget EdAsmGadget = {
        &PrtAsmGadget,
        92, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &EdAsmText,
        0, NULL, 0, NULL
    };
/* As Gadget */
    struct IntuiText AsText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"   as   ",
        NULL,
    };
    struct Gadget AsGadget = {
        &EdAsmGadget,
        171, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &AsText,
        0, NULL, 0, NULL
    };
/* AsLn Gadget */
    struct IntuiText AsLnText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)" as  ln ",
        NULL,
    };
    struct Gadget AsLnGadget = {
        &AsGadget,
        250, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &AsLnText,
        0, NULL, 0, NULL
    };
/* LoadOpt Gadget */
    struct IntuiText LoadOptText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"LoadOpt.",
        NULL,
    };
    struct Gadget LoadOptGadget = {
        &AsLnGadget,
        487, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &LoadOptText,
        0, NULL, 0, NULL
    };
/* Load File Gadget */
    struct IntuiText LoadFileText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"LoadFile",
        NULL,
    };
    struct Gadget LoadFileGadget = {
        &LoadOptGadget,
        408, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &LoadFileText,
        0, NULL, 0, NULL
    };
/* PrtErrDat Gadget */
    struct IntuiText PrtErrDatText = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"Prt Err ",
        NULL,
    };
    struct Gadget PrtErrDatGadget = {
        &LoadFileGadget,
        329, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &PrtErrDatText,
        0, NULL, 0, NULL
    };
/* Dir DF0: Gadget */
    struct IntuiText DirDf0Text = {
        GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
        (UBYTE *)"Dir DF0:",
        NULL,
    };
    struct Gadget DirDf0Gadget = {
        &PrtErrDatGadget,
        566, 28, 64, 8,
        GADGHCOMP,
        RELVERIFY | GADGIMMEDIATE,
        BOOLGADGET,
        (APTR)&GadgBorder,
        NULL,
        &DirDf0Text,
        0, NULL, 0, NULL
    };
```

```
Uz0 /* Prt.h Gadget */
Rk3    struct IntuiText PrtHText = {
9x6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
dT        (UBYTE *)" Prt.h   ",
Sh        NULL,
B73    };
01     struct Gadget PrtHGadget = {
2N6       &DirDf0Gadget,
sg        13, 42, 64, 8,
TJ        GADGHCOMP,
Ct        RELVERIFY | GADGIMMEDIATE,
Qr        BOOLGADGET,
5C        (APTR)&GadgBorder,
bq        NULL,
61        &PrtHText,
9A        0, NULL, 0, NULL
MI3    };
sQ0 /* Ed.h Gadget */
fX3    struct IntuiText EdHText = {
QE6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
7W        (UBYTE *)" Ed.h    ",
jy        NULL,
S03    };
w7     struct Gadget EdHGadget = {
766       &PrtHGadget,
MH        92, 42, 64, 8,
ka        GADGHCOMP,
TA        RELVERIFY | GADGIMMEDIATE,
h8        BOOLGADGET,
MT        (APTR)&GadgBorder,
s7        NULL,
nx        &EdHText,
QR        0, NULL, 0, NULL
dZ3    };
OK0 /* Ln Gadget */
C03    struct IntuiText LnText = {
hV6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
n7        (UBYTE *)"   ln    ",
0F        NULL,
jf3    };
g1     struct Gadget LnGadget = {
sy6       &EdHGadget,
wZ        171, 42, 64, 8,
1r        GADGHCOMP,
kR        RELVERIFY | GADGIMMEDIATE,
yP        BOOLGADGET,
dk        (APTR)&GadgBorder,
90        NULL,
Tt        &LnText,
hi        0, NULL, 0, NULL
uq3    };
rZ0 /* CcAsLn Gadget */
wx3    struct IntuiText CcAsLnText = {
ym6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
mF        (UBYTE *)"cc as ln",
HW        NULL,
0w3    };
MT     struct Gadget CcAsLnGadget = {
Xy6       &LnGadget,
5g        250, 42, 64, 8,
I8        GADGHCOMP,
1i        RELVERIFY | GADGIMMEDIATE,
Fg        BOOLGADGET,
u1        (APTR)&GadgBorder,
Qf        NULL,
fN        &CcAsLnText,
yz        0, NULL, 0, NULL
B73    };
me0 /* ClearScr Gadget */
Va3    struct IntuiText ClearScrText = {
F36       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
HI        (UBYTE *)"ClearScr",
Yn        NULL,
HD3    };
6U     struct Gadget ClearScrGadget = {
SR6       &CcAsLnGadget,
1o        487, 42, 64, 8,
ZP        GADGHCOMP,
Iz        RELVERIFY | GADGIMMEDIATE,
Wx        BOOLGADGET,
BI        (APTR)&GadgBorder,
hw        NULL,
in        &ClearScrText,
FG        0, NULL, 0, NULL
S03    };
zF0 /* Start Pr Gadget */
fe3    struct IntuiText StartPrText = {
WK6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
CY        (UBYTE *)"Start Pr",
p4        NULL,
YU3    };
so     struct Gadget StartPrGadget = {
no6       &ClearScrGadget,
ye        408, 42, 64, 8,
qg        GADGHCOMP,
ZG        RELVERIFY | GADGIMMEDIATE,
nE        BOOLGADGET,
SZ        (APTR)&GadgBorder,
yD        NULL,
WM        &StartPrText,
WX        0, NULL, 0, NULL
jf3    };
AG0 /* EdErrDat Gadget */
3y3    struct IntuiText EdErrDatText = {
nb6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
af        (UBYTE *)" Ed Err ",
6L        NULL,
p13    };
iZ     struct Gadget EdErrDatGadget = {
aR6       &StartPrGadget,
N5        329, 42, 64, 8,
7x        GADGHCOMP,
qX        RELVERIFY | GADGIMMEDIATE,
4V        BOOLGADGET,
jq        (APTR)&GadgBorder,
FU        NULL,
DL        &EdErrDatText,
no        0, NULL, 0, NULL
0w3    };
0c0 /* Dir DF1: Gadget */
8m3    struct IntuiText DirDf1Text = {
4s6       GTFRONTPEN, GTBACKPEN, JAM2, 0, 0, &Font,
gs        (UBYTE *)"Dir DF1:",
Nc        NULL,
623    };
Ho     struct Gadget DirDf1Gadget = {
IM6       &EdErrDatGadget,
iT        566, 42, 64, 8,
0E        GADGHCOMP,
7o        RELVERIFY | GADGIMMEDIATE,
Lm        BOOLGADGET,
07        (APTR)&GadgBorder,
W1        NULL,
UZ        &DirDf1Text,
45        0, NULL, 0, NULL
HD3    };
IQ0 UBYTE UndoBuffer[MAXSTRSIZE];
7n  /* StringBorder Definition */
q93    SHORT StringVectors[] = { 0, 0,  227, 0,  227, 11,  0, 1
       1,  0, 1 };
c3     struct Border StringBorder = {
Ky6       -3, -2, SBFRONTPEN, SBBACKPEN, JAM1, 5,
nU        (SHORT *)StringVectors,
WX        NULL
PL3    };
G00 /* Workfile Gadget */
7f3    UBYTE WorkfileBuffer[MAXSTRSIZE];
Gu     struct StringInfo WorkfileInfo = {
Cs6       WorkfileBuffer,
FB        UndoBuffer,
Ik        0,
03        MAXSTRSIZE,
NO        0, 0, 0, 0, 0, 0, NULL, 0, NULL
YU3    };
HC     struct IntuiText WorkfileText = {
7L6       STFRONTPEN, STBACKPEN, JAM1, -77, 0, &Font,
Zc        (UBYTE *)"Workfile:",
kl        NULL
```

Listing 1. (Fortsetzung)

```
dZ3    };
aQ     struct Gadget WorkfileGadget = {
Xt6       &DirDf1Gadget,
kq        92, 56, 220, 10,
vl        GADGHCOMP,
OD        RELVERIFY,
wm        STRGADGET,
GY        (APTR)&StringBorder,
3I        NULL,
SZ        &WorkfileText,
a2        0,
00        (APTR)&WorkfileInfo,
OC        0, NULL
qm3    };
Om0 /* Mainfile Gadget */
jY3    UBYTE MainfileBuffer[MAXSTRSIZE];
Og     struct StringInfo MainfileInfo = {
IU6       MainfileBuffer,
gc        UndoBuffer,
jB        0,
pU        MAXSTRSIZE,
op        0, 0, 0, 0, 0, 0, NULL, 0, NULL
zv3    };
AK     struct IntuiText MainfileText = {
Ym6       STFRONTPEN, STBACKPEN, JAM1, -77, 0, &Font,
fE        (UBYTE *)"Mainfile:",
BC        NULL
403    };
MW     struct Gadget MainfileGadget = {
hk6       &WorkfileGadget,
B3        92, 70, 220, 10,
MC        GADGHCOMP,
pe        RELVERIFY,
ND        STRGADGET,
hz        (APTR)&StringBorder,
Uj        NULL,
4S        &MainfileText,
1T        0,
HW        (APTR)&MainfileInfo,
Rd        0, NULL
HD3    };
Tp0 /* Inclfile Gadget */
883    UBYTE InclfileBuffer[MAXSTRSIZE];
Tj     struct StringInfo InclfileInfo = {
gt6       InclfileBuffer,
73        UndoBuffer,
Ac        0,
Gv        MAXSTRSIZE,
FG        0, 0, 0, 0, 0, 0, NULL, 0, NULL
QM3    };
kP     struct IntuiText InclfileText = {
zD6       STFRONTPEN, STBACKPEN, JAM1, -77, 0, &Font,
3d        (UBYTE *)"Inclfile:",
cd        NULL
VR3    };
Pu     struct Gadget InclfileGadget = {
Jd6       &MainfileGadget,
qi        92, 84, 220, 10,
nd        GADGHCOMP,
G5        RELVERIFY,
oe        STRGADGET,
8Q        (APTR)&StringBorder,
vA        NULL,
T2        &InclfileText,
Su        0,
rb        (APTR)&InclfileInfo,
s4        0, NULL
ie3    };
eE0 /* Compiler Options Gadget */
X03    UBYTE CompOptBuffer[MAXSTRSIZE] = {"-a"};
PD     struct StringInfo CompOptInfo = {
ir6       CompOptBuffer,
YU        UndoBuffer,
b3        0,
hM        MAXSTRSIZE,
gh        0, 0, 0, 0, 0, 0, NULL, 0, NULL
rn3    };
9w     struct IntuiText CompOptText = {
Ti6       STFRONTPEN, STBACKPEN, JAM1, -78, 0, &Font,
Z6        (UBYTE *)"CompOpt:",
34        NULL
ws3    };
Bb     struct Gadget CompOptGadget = {
iD6       &InclfileGadget,
kj        408, 56, 220, 10,
E4        GADGHCOMP,
hW        RELVERIFY,
F5        STRGADGET,
Zr        (APTR)&StringBorder,
Mb        NULL,
4q        &CompOptText,
tL        0,
BF        (APTR)&CompOptInfo,
JV        0, NULL
953    };
Ya0 /* Assembler Options Gadget */
kA3    UBYTE AssmOptBuffer[MAXSTRSIZE] = {""};
7g     struct StringInfo AssmOptInfo = {
Qe6       AssmOptBuffer,
zv        UndoBuffer,
2U        0,
8n        MAXSTRSIZE,
78        0, 0, 0, 0, 0, 0, NULL, 0, NULL
IE3    };
Ge     struct IntuiText AssmOptText = {
u96       STFRONTPEN, STBACKPEN, JAM1, -78, 0, &Font,
Ht        (UBYTE *)"AssmOpt:",
UV        NULL
NJ3    };
eJ     struct Gadget AssmOptGadget = {
I56       &CompOptGadget,
xs        408, 70, 220, 10,
fV        GADGHCOMP,
8x        RELVERIFY,
gW        STRGADGET,
0I        (APTR)&StringBorder,
n2        NULL,
rx        &AssmOptText,
Km        0,
Ix        (APTR)&AssmOptInfo,
kw        0, NULL
aW3    };
CE0 /* Link Options Gadget */
dT3    UBYTE LinkOptBuffer[MAXSTRSIZE] = {"-lc"};
DU     struct StringInfo LinkOptInfo = {
FN6       LinkOptBuffer,
QM        UndoBuffer,
Tv        0,
ZE        MAXSTRSIZE,
YZ        0, 0, 0, 0, 0, 0, NULL, 0, NULL
jf3    };
3k     struct IntuiText LinkOptText = {
La6       STFRONTPEN, STBACKPEN, JAM1, -78, 0, &Font,
6c        (UBYTE *)"LinkOpt:",
vw        NULL
ok3    };
S8     struct Gadget LinkOptGadget = {
5C6       &AssmOptGadget,
cc        408, 84, 220, 10,
6w        GADGHCOMP,
Z0        RELVERIFY,
7x        STRGADGET,
Rj        (APTR)&StringBorder,
ET        NULL,
ak        &LinkOptText,
lD        0,
53        (APTR)&LinkOptInfo,
BN        0, NULL
1x3    };
400 /* Window Definition */
xu3    struct NewWindow NewWindow = {
Q96       0, 0, 640, 96, WDETAILPEN, WBLOCKPEN,
ZX        CLOSEWINDOW | GADGETUP,
z5        WINDOWCLOSE | SMART_REFRESH | ACTIVATE | WINDOWDRAG |
           WINDOWDEPTH | NOCAREREFRESH,
ep        &LinkOptGadget,
vN        0,
tU        (UBYTE *)"Aztec-C Control-Window V4.0", 0, 0,
tR        0, 0, 0, 0,
p8        WBENCHSCREEN
C83    };
(C) 1989 M&T
```

Listing 2. (Schluß)

Programmname:	Listing 2
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec
Aufrufe:	siehe Text

```
Programmautor: Hans-Jrgen Pierick
----------------------------------

/* äöü Drucker umschalten */
/*****************************************************************/
/*                    Aztec-C Control-Window V4.0                */
/*****************************************************************/
/*    Programmname: Cont.c          *                            */
/* Includefilename: ContWind.h      *               H.J.P.       */
/*           Datum: Oct./Nov. 88    *                            */
/*****************************************************************/
#define MAXSTRSIZE 200        /* UndoBuffer für StringGadgets */
#define EDITOR    "ed "       /* Ein Space hinter Editornamen */
#define GBFRONTPEN   3        /* Gadget Border Front Farbe */
#define GBBACKPEN    0        /* Gadget Border Back  Farbe */
#define GTFRONTPEN   3        /* Gadget Text   Front Farbe */
#define GTBACKPEN    2        /* Gadget Text   Back  Farbe */
#define SBFRONTPEN   2        /* String Border Front Farbe */
#define SBBACKPEN    2        /* String Border Back  Farbe */
#define STFRONTPEN   3        /* String Text   Front Farbe */
#define STBACKPEN    0        /* String Text   Back  Farbe */
#define WDETAILPEN   2        /* Window Text         Farbe */
#define WBLOCKPEN    3        /* Window Border       Farbe */
#include <stdio.h>
#include <exec/types.h>
#include <intuition/intuition.h>
#include <functions.h>
#include "ContWind.h"
FILE *fp, *fopen();
static RamDisk=0;
static ErrDat=0;
static UBYTE exstr[MAXSTRSIZE];
main()
   {
     struct Window *Window;
     struct IntuiMessage *Message;
     ULONG class;
     APTR address;
     if ((IntuitionBase=OpenLibrary("intuition.library", 33L)) == NULL)
        goto beenden;
     if ((GfxBase=OpenLibrary("graphics.library", 33L)) == NULL)
        goto beenden;
     if ((Window=OpenWindow(&NewWindow)) == NULL)
        goto beenden;
     SwitchGadgets(OffGadget, Window);
     ActivateGadget(&MainfileGadget, Window, NULL);
     FOREVER
       {
        Wait(1L<<Window->UserPort->mp_SigBit);
        while (Message=(struct IntuiMessage *)GetMsg(Window->UserPort))
          {
           class=Message->Class;
           address=Message->IAddress;
           ReplyMsg(Message);
           switch(class)
             {
              case GADGETUP:
                GadgetMessage(address,Window);
                break;
              case CLOSEWINDOW:
                while (Message=(struct IntuiMessage *)GetMsg(Window->UserPort)) ReplyMsg(Message);
                goto beenden;
             } /* switch */
          } /* while */
       } /* FOREVER */
beenden:
     if (Window)         CloseWindow(Window);
     if (GfxBase)        CloseLibrary(GfxBase);
     if (IntuitionBase)  CloseLibrary(IntuitionBase);
   } /* main */
GadgetMessage(address,w)
   APTR address;
   struct Window *w;
   {
/* RAM Disk */
    if (address == (APTR)&RamDiskGadget) RamDisk ? (RamDisk=0) : (RamDisk=1);
/* ErrDat */
    if (address == (APTR)&ErrDatGadget) ErrDat ? (ErrDat=0) : (ErrDat=1);
/* Mainfile */
    if ((address == (APTR)&MainfileGadget) && (MainfileBuffer[0]))
      {
       strcpy(&WorkfileBuffer, &MainfileBuffer);
       strcat(WorkfileBuffer, ".c");
       RefreshGadgets(&WorkfileGadget, w, NULL);
       ActivateGadget(&InclfileGadget, w, NULL);
       SwitchGadgets(OnGadget, w);
      } /* if */
/* Save Opt */
    if (address == (APTR)&SaveOptGadget)
      {
       fp=fopen("ContDir/Cont.opt", "w");
       if (fp != NULL)
         {
          fwrite(CompOptBuffer, MAXSTRSIZE, 1, fp);
          fwrite(LinkOptBuffer, MAXSTRSIZE, 1, fp);
          fwrite(AssmOptBuffer, MAXSTRSIZE, 1, fp);
          fclose(fp);
         } /* if */
      } /* if */
/* Load Opt */
    if (address == (APTR)&LoadOptGadget)
      {
       fp=fopen("ContDir/Cont.opt", "r");
       if (fp != NULL)
         {
          fread(CompOptBuffer, MAXSTRSIZE, 1, fp);
          fread(LinkOptBuffer, MAXSTRSIZE, 1, fp);
          fread(AssmOptBuffer, MAXSTRSIZE, 1, fp);
          fclose(fp);
          RefreshGadgets(&LinkOptGadget, w, NULL);
          RefreshGadgets(&AssmOptGadget, w, NULL);
          RefreshGadgets(&CompOptGadget, w, NULL);
         } /* if */
      } /* if */
/* Save File */
    if (address == (APTR)&SaveFileGadget)
      {
       fp=fopen("ContDir/Cont.file", "w");
       if (fp != NULL)
         {
          fwrite(WorkfileBuffer, MAXSTRSIZE, 1, fp);
          fwrite(MainfileBuffer, MAXSTRSIZE, 1, fp);
          fwrite(InclfileBuffer, MAXSTRSIZE, 1, fp);
          fclose(fp);
         } /* if */
      } /* if */
/* Load File */
    if (address == (APTR)&LoadFileGadget)
```

Listing 2. Mit »Cont.c« steuern Sie den Aztec per Mausklick

```
yR6         {
iF7         fp=fopen("ContDir/Cont.file", "r");
We          if (fp != NULL)
1U9           {
CgA           fread(WorkfileBuffer, MAXSTRSIZE, 1, fp);
09            fread(MainfileBuffer, MAXSTRSIZE, 1, fp);
NJ            fread(InclfileBuffer, MAXSTRSIZE, 1, fp);
H9            fclose(fp);
bG            RefreshGadgets(&WorkfileGadget, w, NULL);
nj            RefreshGadgets(&MainfileGadget, w, NULL);
mt            RefreshGadgets(&InclfileGadget, w, NULL);
fR            SwitchGadgets(OnGadget, w);
uu9           }  /* if */
vv6         }  /* if */
kr0 /* Ed.c */
Nr4     if (address == (APTR)&EdCGadget)
Eh6       {
ik7       SwitchGadgets(OffGadget, w);
2X        ExEd(WorkfileBuffer);
nZ        SwitchGadgets(OnGadget, w);
wk6       }  /*  if */
DN0 /* Start Pr */
Oq4     if (address == (APTR)&StartPrGadget)
Lo6       {
pr7       SwitchGadgets(OffGadget, w);
yv        Execute(MainfileBuffer,0L,0L);
ug        SwitchGadgets(OnGadget, w);
3r6       }  /*  if */
bL0 /* AsLn */
u54     if (address == (APTR)&AsLnGadget)
Sv6       {
wy7       SwitchGadgets(OffGadget, w);
Ut        As();
c7        Ln();
2o        SwitchGadgets(OnGadget, w);
HH6       }  /* if */
920 /* CcAsLn */
zS4     if (address == (APTR)&CcAsLnGadget)
a36       {
467       SwitchGadgets(OffGadget, w);
u5        Cc();
d2        As();
1G        Ln();
Bx        SwitchGadgets(OnGadget, w);
QQ6       }  /* if */
UX0 /* Cc */
f04     if (address == (APTR)&CcGadget)
jC6       {
DF7       SwitchGadgets(OffGadget, w);
3E        Cc();
I4        SwitchGadgets(OnGadget, w);
XX6       }  /* if */
zy0 /* As */
wV4     if (address == (APTR)&AsGadget)
qJ6       {
KM7       SwitchGadgets(OffGadget, w);
sH        As();
PB        SwitchGadgets(OnGadget, w);
ee6       }  /* if */
Vz0 /* Ln */
M14     if (address == (APTR)&LnGadget)
xQ6       {
RT7       SwitchGadgets(OffGadget, w);
6b        Ln();
WI        SwitchGadgets(OnGadget, w);
116       }  /* if */
Jt0 /* Dir DF0: */
Va4     if (address == (APTR)&DirDf0Gadget)
4X6       {
NF7       ExDir("df0:");
qq6       }  /* if */
U00 /* Dir DF1: */
ek4     if (address == (APTR)&DirDf1Gadget)
9c6       {
YM7       ExDir("df1:");
vv6       }  /* if */
md0 /* Dir RAM: */
o14     if (address == (APTR)&DirRamGadget)
Eh6       {
w17       ExDir("ram:");
006       }  /* if */
HQ0 /* Prt.c */
8G4     if (address == (APTR)&PrtCGadget)
Jm6       {
np7       SwitchGadgets(OffGadget, w);
1e        ExPrint(WorkfileBuffer, "");
se        SwitchGadgets(OnGadget, w);
776       }  /* if */
Ph0 /* Prt.asm */
1S4     if (address == (APTR)&PrtAsmGadget)
Qt6       {
uw7       SwitchGadgets(OffGadget, w);
XV        ExPrint(MainfileBuffer, ".asm");
zl        SwitchGadgets(OnGadget, w);
EE6       }  /* if */
p30 /* Prt.h */
Wj4     if (address == (APTR)&PrtHGadget)
X06       {
137       SwitchGadgets(OffGadget, w);
ap        ExPrint(InclfileBuffer, "");
6s        SwitchGadgets(OnGadget, w);
LL6       }  /* if */
gL0 /* Ed .asm */
m34     if (address == (APTR)&EdAsmGadget)
e76       {
8A7       SwitchGadgets(OffGadget, w);
57        ExBefehl("ed ", ".asm", "");
Dz        SwitchGadgets(OnGadget, w);
SS6       }  /* if */
hT0 /* ClearScr */
Pm4     if (address == (APTR)&ClearScrGadget)
1E6       {
MH7       exstr[0]='\0';
0q        strcat(exstr, "cls");
c0        Execute(exstr, 0L, 0L);
ZZ6       }  /* if */
dp0 /* Ed.h */
Vf4     if (address == (APTR)&EdHGadget)
sL6       {
M07       SwitchGadgets(OffGadget, w);
IK        ExEd(InclfileBuffer);
RD        SwitchGadgets(OnGadget, w);
gg6       }  /* if */
tA0 /* Prt Err */
2a4     if (address == (APTR)&PrtErrDatGadget)
zS6       {
TV7       SwitchGadgets(OffGadget, w);
Cg        ExPrint("ContDir/Cont.err", "");
YK        SwitchGadgets(OnGadget, w);
nn6       }  /* if */
Pf0 /* Ed Err */
kx4     if (address == (APTR)&EdErrDatGadget)
6Z6       {
ac7       SwitchGadgets(OffGadget, w);
mU        ExEd("ContDir/Cont.err");
fR        SwitchGadgets(OnGadget, w);
uu6       }  /* if */
C33   }  /* Gadget Message */
sn0 Cc()
Dg3   {
2B4   UBYTE txt[25];
Rg    (ErrDat) ? (strcpy(txt, "cc >ContDir/Cont.err ")) : (strcpy(txt, "cc "));
TJ    if (RamDisk)
Hk6     {
097     ExBefehl(txt, " -O RAM:cprtemp.asm ", CompOptBuffer);
336     }  /* if */
D04   else
Lo6     {
6a7     ExBefehl(txt, " ", CompOptBuffer);
gP6     }  /* else */
6R3   }  /* Cc */
nw0 As()
Qt3   {
eu4   if (RamDisk)
Sv6     {
a77     Execute("as RAM:cprtemp.asm", 0L, 0L);
FL      Execute("delete RAM:cprtemp.asm", 0L, 0L);
FF6     }  /* if */
PC4   else
X06     {
E57     ExBefehl("as ", " ", AssmOptBuffer);
sb6     }  /* else */
En3   }  /* As */
6L0 Ln()
```

```
c53    {
q64     if (RamDisk)
e76      {
2Q7       ExBefehl("ln -O ", " RAM:cprtemp.o ", LinkOptBuffer)
          ;
1j        Execute("delete RAM:cprtemp.o", OL, OL);
RR6      }  /* if */
b04     else
jC6      {
3w7       ExBefehl("ln ", ".o ", LinkOptBuffer);
4n6      }  /* else */
dI3    }  /* Ln */
hk0 ExBefehl(str1, str2, str3)
ny3    UBYTE *str1[], *str2[], *str3[];
pI     {
QL4     exstr[0]='\0';
vM      strcat(exstr, str1);
MV      strcat(exstr, MainfileBuffer);
1T      strcat(exstr, str2);
6Z      strcat(exstr, str3);
j7      Execute(exstr, OL, OL);
Uj3    }  /* ExBefehl */
0c0 ExPrint(str1, str2)
Ap3    UBYTE *str1[], str2[];
zS     {
aV4     exstr[0]='\0';
7E      strcat(exstr, "type ");
6X      strcat(exstr, str1);
Bd      strcat(exstr, str2);
N8      strcat(exstr, " to prt:");
tH      Execute(exstr, OL, OL);
cb3    }  /* ExPrint */
1h0 ExDir(str1)
433    UBYTE *str1[];
9c     {
kf4     exstr[0]='\0';
6S      strcat(exstr, "dir ");
Gh      strcat(exstr, str1);
1P      Execute(exstr, OL, OL);
P73    }  /* ExDir */
ga0 ExEd(str1)
CB3    UBYTE *str1[];
Hk     {
sn4     exstr[0]='\0';
Xm      strcat(exstr, EDITOR);
Op      strcat(exstr, str1);
9X      Execute(exstr, OL, OL);
5o3    }  /* ExEd */
```

```
KJ0 SwitchGadgets(funcp, w)
u63    void (*funcp)();
Ff     struct Window *w;
Qt     {
LV4     (*funcp)(&PrtCGadget,        w, NULL);
Ob      (*funcp)(&PrtAsmGadget,      w, NULL);
m1      (*funcp)(&PrtHGadget,        w, NULL);
07      (*funcp)(&EdCGadget,         w, NULL);
4A      (*funcp)(&EdAsmGadget,       w, NULL);
MY      (*funcp)(&EdHGadget,         w, NULL);
ao      (*funcp)(&CcGadget,          w, NULL);
J1      (*funcp)(&AsGadget,          w, NULL);
Rz      (*funcp)(&LnGadget,          w, NULL);
wv      (*funcp)(&AsLnGadget,        w, NULL);
Of      (*funcp)(&CcAsLnGadget,      w, NULL);
yB      (*funcp)(&PrtErrDatGadget,   w, NULL);
Y1      (*funcp)(&EdErrDatGadget,    w, NULL);
1j3    }  /* SwitchGadgets */
(C) 1989 M&T
```

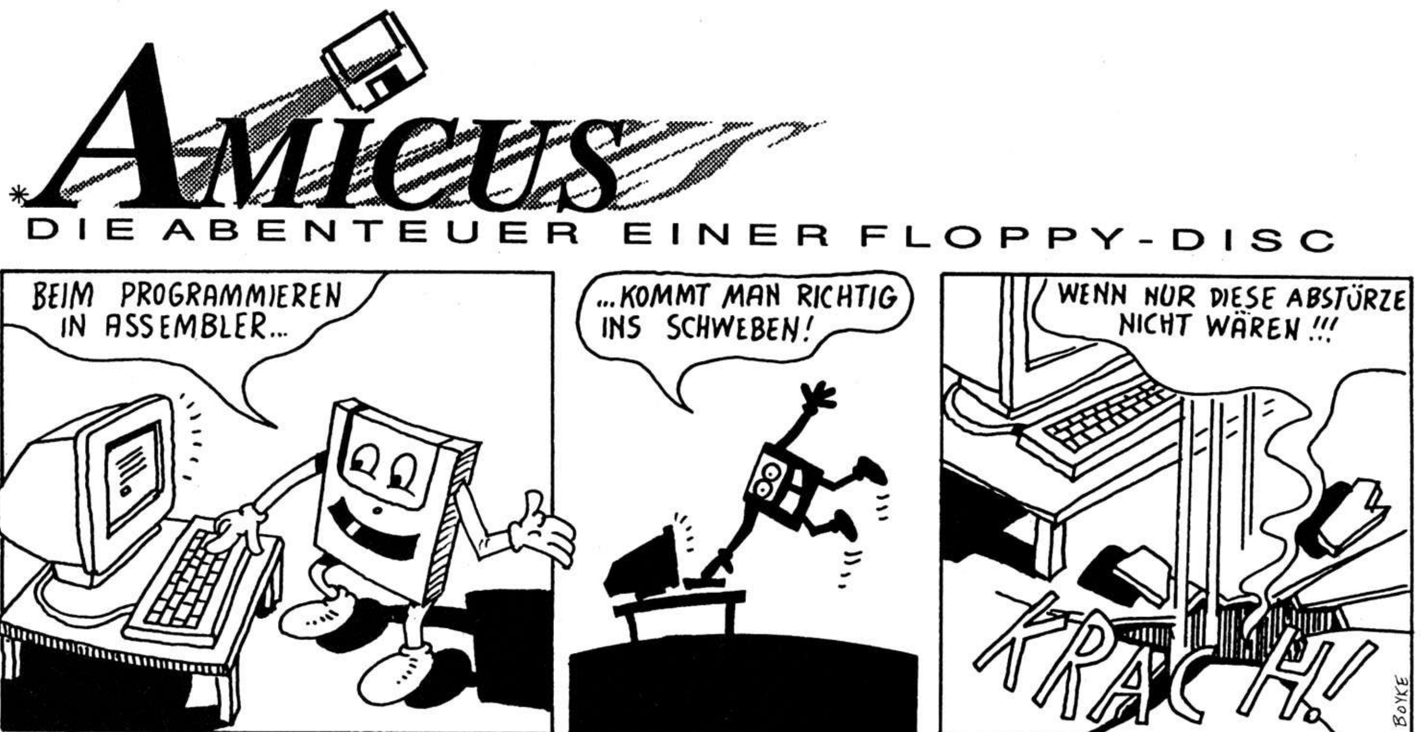
Listing 2. (Schluß)

Programmname:	Listing 3
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec
Aufrufe:	siehe Text

```
Programmautor: Hans_Jürgen Pierick
----------------------------------

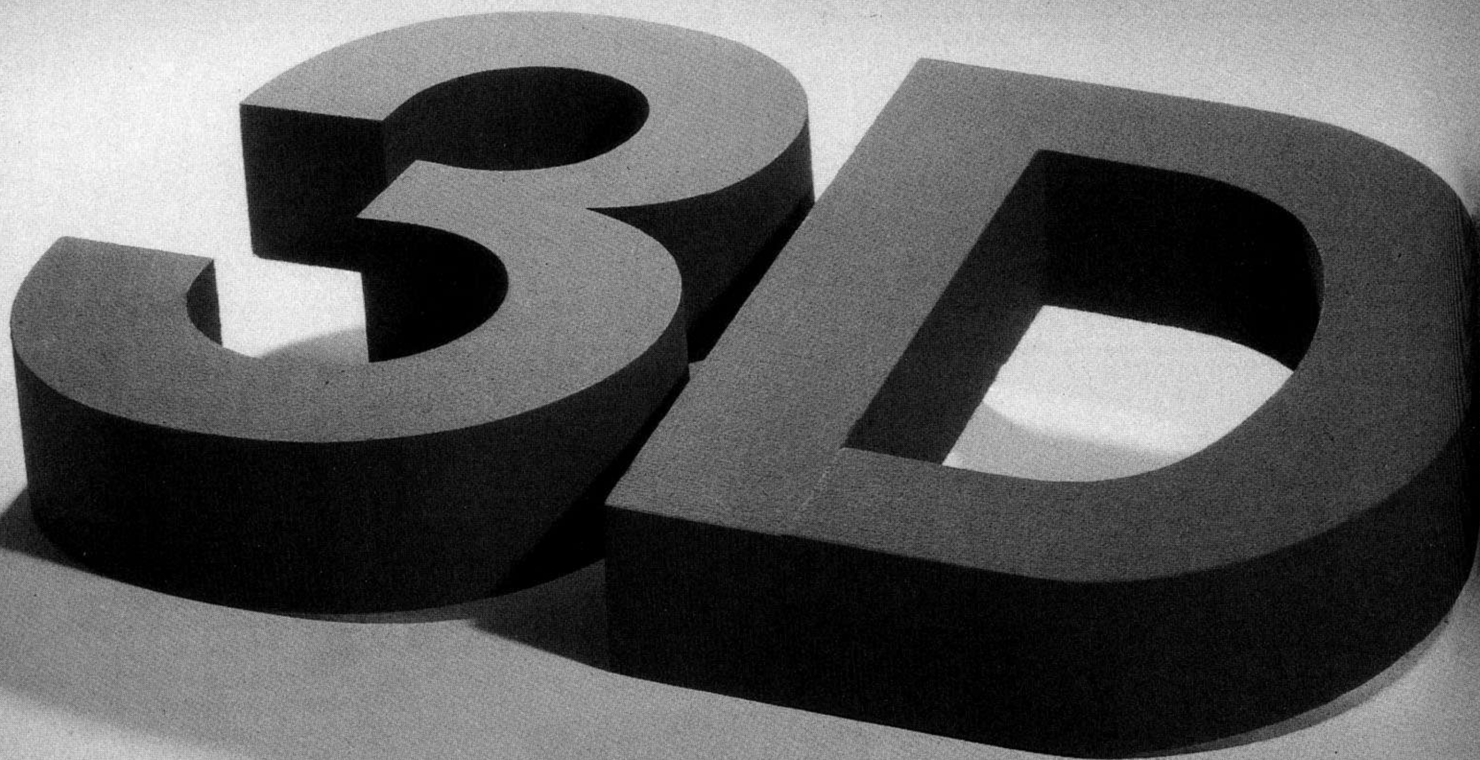
ds0 /************************
8y  **                      **
tm  **  CLS.c fuehrt einen  **
Ca  **  Seitenvorschub aus  **
B1  **                      **
ix  ************************/
oA  main()
4X  {
SR3    printf("\f");
Af0 }
(C) 1989 M&T
```

Listing 3. »CLS.c« löscht das aktuelle CLI-Fenster



Grafik in der dritten Dimension

»Ray Tracer« heißt wörtlich übersetzt »Strahlenverfolger«. Wer aber glaubt, mit dem folgenden Programm könne er seinen Amiga als Geigerzähler einsetzen, der irrt sich. Die Strahlen sind in unserem Fall harmloses Licht und verfolgt werden sie zur dreidimensionalen Darstellung.



Ray Tracing« nennt man die perspektivische Darstellung von dreidimensionalen Objekten unter Berücksichtigung der Lichtstrahlen. Was in diesem kurzen Satz an List und Tücke für den Programmierer steckt, wird erst vollends klar, wenn Sie sich verdeutlichen, was dabei berücksichtigt werden muß.

So müssen nicht nur die Abmessungen der Objekte (in unserem Fall Kugeln) der Perspektive entsprechend umgerechnet werden. Es muß auch der Lichteinfall berücksichtigt, die Spiegelungen richtig dargestellt (möglicherweise auf einer gekrümmten Oberfläche) und eventuell einzelne Flächen durchsichtig gemacht werden.

Bei weniger leistungsfähigen Computern erweisen sich derartige Programme oft genug als wahre »Rechenzeitkiller«, die den Prozessor für erhebliche Zeit lahmlegen. Aber auch der Amiga – mit 68000er und dem Customchip Blitter sehr gut gerüstet – ist mit dieser Aufgabe stark gefordert.

So kann die Berechnung einer Grafik mit mehreren Elementen im Interlace-Modus durchaus zwei bis drei Stunden in Anspruch nehmen. Das heißt aber nicht, daß Sie sich einen zweiten Amiga kaufen müssen, weil Ihrer mit Raytracing demnächst ausgelastet ist. Es gibt einige Möglichkeiten, die Ausgabe zu beschleunigen oder notfalls abzubrechen.

Solange Sie an Ihrer Grafik noch Änderungen vornehmen, sollten Sie grundsätzlich im »Low-Resolution«-Modus arbeiten. Dieser ist etwa viermal so schnell wie der Interlace-Modus. Eine erhebliche Verkürzung der Rechenzeit bietet die Berechnung im »Test«-Modus. Diese stellt die Grafik im Briefmarkenformat dar. Sie brauchen auf eine Grafik nur wenige Minuten warten, für die sonst einige Stunden nötig wären.

Wollen Sie weitere Rechenzeit einsparen, ziehen Sie einfach während des Bildschirmaufbaus den Screen nach unten (oder stellen im Menü »verdeckt« ein). Sie müssen allerdings darauf achten, den Screen wieder ganz nach oben zu ziehen, bevor Sie mit »Window ändern« die Einstellungen bearbeiten. Denn dieser Menüpunkt öffnet ein eigenes Fenster, das sich nur durch ein Gadget am unteren Rand beseitigen läßt.

Sollten Sie Ihren Amiga zwischendurch für dringende andere Aufgaben benötigen, so können Sie die Berechnung mit einem einfachen Mausklick an einer beliebigen Stelle des Screens abbrechen. Speichern Sie jetzt den derzeitigen Stand der Grafik. Wenn Sie weitermachen wollen, laden Sie diese und arbeiten an derselben Stelle weiter.

Leider ist es uns aufgrund der Länge dieses Programms nicht möglich, das Listing abzudrucken. Der Raytracer ist nur auf unserer Programmservice-Diskette enthalten, die Sie mit der Karte auf Seite 34 bestellen können. Dort finden Sie selbstverständlich auch den dokumentierten Sourcecode zu diesem Programm.

Starten Sie den Raytracer entweder vom CLI mit »rt« und <RETURN> oder von der Workbench per Doppelklick auf das Icon. Sie sehen jetzt einen neuen Screen mit der Titelleiste. Drücken Sie die rechte

Programm-Service

Maustaste, dann sehen Sie vier Menüs: »RayTracer«, »Auflösung«, »Berechnen« und »Extras«:

Auflösung: Wählen Sie den Punkt »High« an, so wird der Bildschirm auf »High-Resolution« und Interlace umgestellt (640 x 512 Punkte).

Bevor Ihnen vom Flackern die Augen schmerzen, schalten Sie lieber zurück (mit dem Menüpunkt »Low«). Der Bildschirm sollte jetzt wieder die normale Auflösung von 320 x 256 Punkten haben.

Berechnen: Hier finden Sie die beiden Punkte »Test« und »Start«. Bitte zügeln Sie vorerst Ihre Neugier und lassen Sie den Punkt »Start« links liegen. Dieser bringt längere Wartezeiten mit sich. Starten Sie statt dessen einen »Test«-Lauf. Dieser berechnet – weil noch keine Objekte definiert wurden – in wenigen Sekunden die Grafik. Sie sehen jetzt in der linken oberen Ecke eine Grafik im Briefmarkenformat, die ein rotgrünes Schachbrettmuster und einen schmalen blauen Streifen als Horizont anzeigt.

```
              -*- Ray Tracer -*-
Bildschirmmitte  X = 0.000000
                 Y = 0.000000
                 Z = 17.999999
Blickrichtung    X = 5.000000
                 Y = 10.000000
                 Z = -2.600000
Bildschirmabstand  = 0.280000
Brechungszahl      = 2.560000
                   »
Kugelnummer [RET]  = 0
                   «
Durchmesser        = 50
Kugelposition    X = -50
                 Y = 200
                 Z = 200
Sp/Gl/Li (1/2/3)   = 3
Kugel löschen (!)    ø

                     OK
```

Bild 1. Mit dem Menüpunkt »W. ändern« stellen Sie alle wichtigen Daten für Ihre Grafik ein

Natürlich ist es nicht Sinn eines Raytracers, eine Flachlandsilhouette mit Schachbrettmuster auf den Bildschirm zu zaubern. Dafür gibt es weit anspruchslosere Programme.

Extras: Gehen wir also daran, das erste Objekt zu definieren. Zu diesem Zweck wählen Sie im Menü »Extras« den Punkt »W. ändern«. Sie sehen ein neues Fenster (Bild 1), in dem Sie alle wichtigen Parameter einstellen.

Sie sehen an der aktuellen Einstellung, daß ein Objekt bereits definiert ist. Es handelt sich dabei um die Kugel mit der Nummer Null, die als Lichtquelle dient. Sie können die Position dieser Lichtquelle ändern, indem Sie mit der Maus zweimal auf die entsprechenden Werte klicken und diese editieren, sobald der Cursor auftaucht.

Ähnlich gehen Sie vor, wenn Sie eine neue Kugel plazieren wollen. Klicken Sie zweimal auf die Nummer der Kugel (im Moment »0«) und tragen Sie dort eine Eins ein. Diese Änderung wird allerdings erst akzeptiert, wenn Sie mit einem Klick auf das Feld »[RET]« bestätigen. Sie sehen jetzt die aktuellen Maße Ihrer neuen Kugel angezeigt, nämlich fünf Nullen. Geben Sie ihr neue Werte, indem Sie die alten wie oben beschrieben ändern. Achten Sie darauf, daß Sie am Anfang Werte eintragen, die im darstellbaren Bereich liegen – später können Sie immer noch experimentieren.

Wir schlagen vor, daß Sie folgende Startwerte eintragen: Durchmesser 50, X-Position 50, Y-Position 100, Z-Position 200 und bei »Sp/Gl/Li« eine 2. Der letzte Wert unterscheidet zwischen spiegelnder (1) und glatter (2) Oberfläche beziehungsweise einer Lichtquelle (3). Klicken Sie mit der Maus jetzt auf »[RET]«, um die neuen Einstellungen zu bestätigen und danach auf das »ok«-Gadget am unteren Bildschirmrand.

Wenn Sie jetzt noch einmal »Test« anwählen, sehen Sie die Grafik mit Ihrer neu definierten

Die erste eigene »Landschaft«

Kugel am rechten Rand. Außerdem dürfte Ihnen aufgefallen sein, daß die Berechnung der Grafik jetzt deutlich mehr Zeit in Anspruch nimmt.

Um einen ersten Eindruck zu erhalten, sollten Sie Ihren Amiga vielleicht für zwei bis drei Stunden »beschäftigen«. Stellen Sie den Interlace-Modus ein (mit »Auflösung«/»High«), starten Sie die Berechnung (mit »Berechnen«/»Start«) und ziehen Sie den Screen ganz nach unten. Der Lohn des Wartens ist eine Grafik, die zwar leicht flackert (wegen Interlace), aber immerhin einen kleinen Eindruck von der Leistungsfähigkeit des Amiga vermittelt. Wenn Sie alles genau befolgt haben, sollte das Bild etwa so aussehen, wie das Foto im Inhaltsverzeichnis, das bei unseren ersten Versuchen entstand.

Wenn Sie nach diesem Vorgeschmack Lust zum Experimentieren bekommen haben, steht dem nichts mehr im Wege. Sehen Sie sich den Menüpunkt »W. ändern« etwas genauer an. Sie finden unter anderem Möglichkeiten, Ihren Blickwinkel (unter »Blickrichtung«), den Abstand vom Ort des Geschehens (mit »Bildschirmabstand«) und die Brechungszahl zu verändern. Experimentieren Sie mit diesen Werten. Wenn Sie eine Einstellung gefunden haben, die Sie vollkommen zufriedenstellt, können Sie diese mit »E. sichern« speichern. Das Programm legt eine Datei »RTWerte« an, aus der beim nächsten Start die Voreinstellungen geholt werden.

RayTracer: Auf ähnliche Art können Sie fertige Grafiken speichern und wieder laden. Die entsprechenden Menüpunkte finden Sie unter »Ray-

tracer« (»Laden« und »Saven«). Beide verzweigen in einen Filerequester, der nur Directories oder selbstgespeicherte Grafiken erkennt. Dabei wird hinter jedem von Ihnen gewählten Namen entweder »-l« (wenn es sich um eine Low-Resolution-Grafik handelt) oder »-h« angehängt. Diese Erweiterung können Sie beim Speichern ebenso wie beim Laden weglassen.

Mit dem dritten Menüpunkt (»Ende«) sollten Sie vorsichtig umgehen. Er respektiert Ihren Wunsch und reagiert prompt, ohne vorher zu fragen, ob Sie nicht noch speichern wollen.

Neben den bereits besprochenen Menüpunkten können Sie im »Extras«-Menü die Titelleiste ein- und ausblenden, ein Countdown-Fenster einrichten, das Sie informiert, wieviele Zeilen noch dargestellt werden müssen, und den aktuellen Screen sichtbar beziehungsweise unsichtbar machen. Auch diese Einstellungen werden in RTWerte gespeichert, so daß sie nicht jedesmal neu definiert werden müssen, sondern automatisch beim Start geladen werden.

Mit den Informationen aus diesem Artikel und etwas Experimentierfreude sollten Sie in der Lage sein, fantastische Grafiken auf den Bildschirm zu zaubern. Wenn Sie mehr über die theoretischen Grundlagen wissen wollen, empfehlen wir das Buch »3-D-Grafik-Programmierung« von Data Becker.

(Arno Gölzer/so)

3-D-Grafik-Programmierung; Jennrich, Massmann, Schulz; Data Becker; 284 Seiten; ISBN 3-89011-174-2; Preis 59 Mark

---

Sounds im Eigenbau

Mit dem »Sound-Effekte-Editor« erzeugen Sie komfortabel alle Geräusche. Vom einfachen akustischen Hinweis auf Eingabefehler bis zum Kanonendonner in Spielen reicht das Spektrum. Der Clou: Die Sounds werden als Quellcode gespeichert und sind so leicht für eigene Programme zu verwenden.

Wenn auf dem Mond riesige Meteore einschlagen und dabei gewaltige Krater hinterlassen, so geschieht das vollkommen lautlos. Schuld daran ist die fehlende Lufthülle. Der Schall ist eine hörbare oder zumindest nachweisbare Schwingung, die sich nur in einem Medium, beispielsweise in der Luft oder im Wasser, verbreitet. Jedes Geräusch hat seine eigene, individuelle Schwingung und kann daher eindeutig identifiziert werden. Aber auch eine — zumindest annähernde — mathematische Beschreibung ist möglich. Wir unterscheiden zwischen vier Grundschwingungen:
- Sinus
- Rechteck
- Dreieck
- Sägezahn

Die Klangfarbe eines Tones machen aber im wesentlichen die sogenannten Oberschwingungen aus. Es handelt sich um mehr oder weniger große Abweichungen von der eigentlichen Wellenform. Der Amiga läßt uns in bezug auf das Aussehen der Wellenform völlig freie Hand. Somit ist es möglich, beliebige Geräusche künstlich zu erzeugen - wir müssen nur die Wellenform entsprechend beschreiben. Diese Aufgabe wird mit dem Sound-Effekte-Editor (SEEd) (Listing 1) zum Kinderspiel.

Nach dem Eingeben des Programms mit dem Checksummer (Seite 159) und dem Compilieren starten Sie SEEd vom CLI aus, indem Sie den Programmnamen eingeben und <RETURN> drücken. Haben Sie ein Programm-Icon für SEEd entworfen, können Sie den Soundgenerator auch von der Workbench aus starten. Wichtig ist in jedem Fall, daß sich im Verzeichnis »Libs« die Mathematik-Libraries befinden (mathtrans.library).

Von der Welle zum Klang

Nach dem Programmstart öffnet sich ein Fenster, das in zwei Felder unterteilt ist (Bild 1):

1. Das Bedienungsfeld belegt das rechte Viertel des Bildschirms. Es besteht aus insgesamt zwölf Reglern (Gadgets). Lesen Sie im folgenden deren Funktion:

Großer Schieberegler links:
Grobeinstellen der Frequenz. Schieben Sie den Regelknopf nach unten, wenn Sie niedrigere Frequenzen einstellen möchten.

Großer Schieberegler rechts:
Feineinstellen der Frequenz. Mit den beiden Frequenzreglern können Sie über 65000 Frequenzen einstellen!

Kleiner Schieberegler links:
Hier stellen Sie ein, wie oft der eingestellte Klang wiederholt werden soll.

Kleiner Schieberegler rechts:
Die Lautstärke.

Null:
Wellenform auf Null stellen. Es erscheint ein waagerechter Strich im Editierfenster.

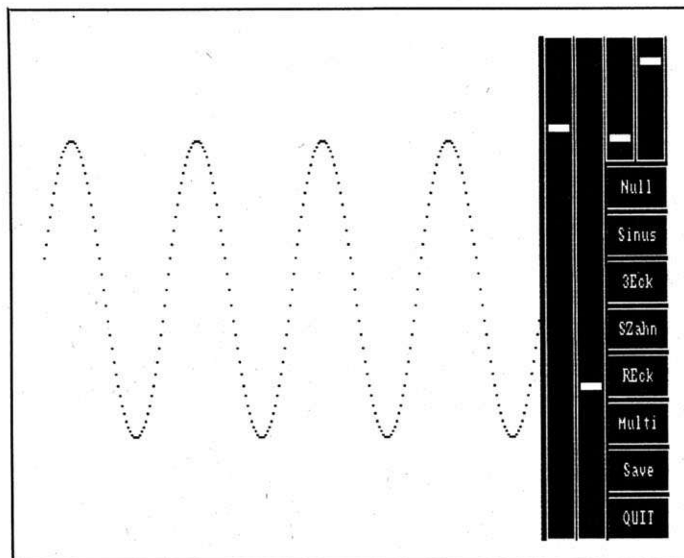


Bild 1. Das SEEd-Fenster. Im linken Teil bearbeiten Sie den gewünschten Sound, rechts befinden sich alle Regler.

Sinus:
Einstellen der Wellenform auf Sinus.
3Eck:
Eine Dreieckschwingung wird eingestellt.
SZahn:
Vorwahl der Sägezahnschwingung.
REck:
Die Rechteckschwingung einstellen.

Die voreingestellten Schwingungen können jetzt beliebig verändert werden (siehe unten). Allerdings ist in diesem Zustand der Klang nicht immer optimal. Verbessern Sie die Qualität, indem Sie mehrere Schwingungen im Zeichenfeld unterbringen:
Multi:
Eingestellte Schwingung verdoppeln.
Save:
»Geräusch« auf Diskette sichern. Ein Requester öffnet sich. Geben Sie den Filenamen an und klicken Sie das Gadget mit der Aufschrift »Save« an (oder »Break«, wenn Sie lieber nicht speichern möchten).
Quit:
Mit diesem Schalter brechen Sie das Programm ab.

Im weiß hinterlegten Zeichenfeld können Sie vorgefertige Schwingungen verändern

Sounds zeichnen mit der Maus

oder aber völlig neue entwerfen. Drücken Sie die linke Maustaste und »zeichnen« Sie Ihren Soundeffekt. Die besten Ergebnisse erhalten Sie, wenn Sie die Höhe des Zeichenfeldes möglichst voll ausnutzen. Im anderen Fall sind die Schwingungen nur schlecht hörbar.

Weiterhin ist es von Vorteil, wenn Schwingungsanfang und -ende auf gleicher Höhe liegen. Nutzen Sie auch hier die Multi-Funktion zur Verbesserung der Soundqualität. Sie hören den Klang, wenn Sie die rechte Maustaste betätigen. Bei den Grundschwingungen ist allerdings selten ein Ton zu hören. Keine Angst, das Programm funktioniert trotzdem fehlerfrei. Verdoppeln Sie die Grundschwingung mehrfach, entlocken Sie dem Amiga bereits einen Klang.

Begrenzen der Wellenform

Mit den orangefarbenen Markierungen im Zeichenfeld können Sie den Anfang und das Ende der Wellenform selbst festlegen. Bedienen Sie die Markierungen mit dem Ziffern-Block der Tastatur:
<7> Startmarkierung nach links,
<9> Startmarkierung nach rechts,
<8> Startmarkierung auf Start.

Da das Datenfeld auf einem Langwort (Longword) beginnen muß, erfolgt die Bewegung der Startmarkierung in 4er-Sprüngen.
<4> Endemarkierung nach links,
<6> Endemarkierung nach rechts,
<5> Endemarkierung auf Ende.

Das Ende des Datenfeldes ist nicht an bestimmte Speicherbereiche gebunden. Daher ist hier noch eine Feineinstellung in Einerschritten möglich:
<1> Endemarkierung einen Schritt nach links,
<3> Endemarkierung einen Schritt nach rechts,
<2> Endemarkierung auf Ende.

Einbindung in C-Programme

Das Einbinden eines SEEd-Sounds in C-Programme ist sehr einfach, da SEEd den Sound als C-Quellcode ablegt. Folgende Schritte sind nötig:
- Binden Sie die SEEd-Datei als Headerfile in Ihr eigenes Programm ein.
- Rufen Sie zu Beginn Ihres eigenen Programms die Funktion SoundOn() auf. Die Funktion liefert im Fehlerfall den Wert 1.

Ihr Sound ist nun initialisiert und Sie können ihn jederzeit und beliebig oft abspielen:
- »Play()« spielt den Soundeffekt ab.
- Rufen Sie vor Beendigung Ihres Programms die Funktion »SoundOff()« auf. Sie gibt alle für den Soundeffekt notwendigen Ressourcen wieder frei.

Play() und SoundOff() liefern keinen Wert.

Hier ein kleines Beispiel:

```
#include "SoundName"

main()
{
        if(!SoundOn()){
            Play();
            SoundOff();
        }
}
```

Aztek-C-Besitzer übersetzen dieses Beispiel so:

```
cc NAME.c +L
ln NAME.o -lc32
```

Anstelle von »NAME« muß natürlich der Name Ihres Programms stehen. Programmierer anderer Programmiersprachen profitieren immer noch von den gesicherten Sounddaten. Deshalb enthält Listing 2 für den Interessierten zum Schluß noch das dokumentierte Listing einer SEEd-Sounddatei. Mit dem SEEd geben wir Ihnen ein wertvolles Werkzeug zur Erzeugung von Sounds in eigenen Programmen in die Hand. Ein Werkzeug, das die Soundprogrammierung ungemein erleichtert und das Sie bestimmt bald nicht mehr missen wollen.

(Arno Gölzer/rs)

Programmname:	SEEd
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztek V3.4
Aufrufe:	cc SEEd.c +L ln SEEd.o -lc32
Bemerkung:	benötigt Math.-Libraries

```
Programmautor: Arno Gölzer
--------------------------

/*
**     Gölzy's SEEd.c
**          08.88
*/
#include <math.h>
#include <stdio.h>
#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfx.h>
#include <devices/audio.h>
#include <exec/memory.h>
#define GH (0xFFFF/255)
#define GL (0xFFFF/255)
#define GC (0xFFFF/64)
#define GV (0xFFFF/64)
#define ENDE (((Ende-Start)/2)+1)
#define MOUSX (win->MouseX)
#define MOUSY (win->MouseY)
#define GAD_ID (((struct Gadget *)ADDR)->GadgetID)
APTR ADDR;
ULONG CLASS, CODE;
UBYTE *Data,buffer=15,name[31],undo[31],Titel[]=(UBYTE *)"-
*- Gölzy's SEEd -*-";
UWORD l,h,p,c,v;
struct Image HDummy,LDummy,CDummy,VDummy;
SHORT Start=0,Ende=510;
SHORT OKpairs[]        ={ 0,0,  61,0,  61,21, 0,21, 0,0 };
SHORT Reqpairs[]       ={ 0,0, 246,0, 246,67, 0,67, 0,0 };
SHORT Spairs[]         ={ 0,0, 210,0, 210,11, 0,11, 0,0 };
struct Border OKRand   ={ -1, -1, 1,0, JAM1, 5,&OKpairs[0],
 NULL       };
struct Border ReqR     ={ 2, 1, 1,0, JAM1, 5,&Reqpairs[0],
NULL       };
struct Border SRand    ={ -1, -2, 1,0, JAM1, 5,&Spairs[0],
NULL       };
struct IntuiText OKT2  ={ 1,0,JAM1,14,6,NULL,(UBYTE *)"QUIT
",NULL      };
struct IntuiText OKText={ 3,0,JAM1,15,7,NULL,(UBYTE *)"QUIT
",&OKT2     };
struct IntuiText SaText={ 1,0,JAM1,14,6,NULL,(UBYTE *)"Save
",NULL      };
struct IntuiText MText ={ 1,0,JAM1,10,6,NULL,(UBYTE *)"Mult
i",NULL     };
struct IntuiText RText ={ 1,0,JAM1,14,6,NULL,(UBYTE *)"REck
",NULL      };
struct IntuiText ZText ={ 1,0,JAM1,10,6,NULL,(UBYTE *)"SZah
n",NULL     };
struct IntuiText DText ={ 1,0,JAM1,14,6,NULL,(UBYTE *)"3Eck
",NULL      };
struct IntuiText SText ={ 1,0,JAM1,10,6,NULL,(UBYTE *)"Sinu
s",NULL     };
struct IntuiText NText ={ 1,0,JAM1,14,6,NULL,(UBYTE *)"Null
```

Listing 1. Mit dem Sound-Effekte-Editor (SEEd) entwerfen Sie Geräusche mit der Maus

```
",NULL      };
struct IntuiText ReqT2 ={ 1,0,JAM1,45,6,NULL,Titel,NULL
            };
struct IntuiText ReqT  ={ 3,0,JAM1,46,7,NULL,Titel,&ReqT2
            };
struct IntuiText RCaT  ={ 1,0,JAM1,10,6,NULL,(UBYTE *)"Brea
k",NULL     };
struct IntuiText RTGT  ={1,0,JAM1,70,-10,NULL,(UBYTE *)"Fil
ename:",NULL};
struct PropInfo PHInfo ={ FREEVERT| AUTOKNOB,0,0,      0,GH,0
,0,0,0,0,0  };
struct PropInfo PLInfo ={ FREEVERT| AUTOKNOB,0,GL*255,0,GL,0
,0,0,0,0,0  };
struct PropInfo CInfo  ={ FREEVERT| AUTOKNOB,0,GC*64, 0,GC,0
,0,0,0,0,0  };
struct PropInfo VInfo  ={ FREEVERT| AUTOKNOB,0,0,      0,GV,0
,0,0,0,0,0  };
struct StringInfo Sinfo={ name,undo,0,31,0,0,0,0,0,0,0,NU
LL          };
struct Gadget RTGad    ={ NULL,20,30,210,10,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,STRGADGET| REQGADGET,
     (APTR)&SRand,NULL,&RTGT,0,(APTR)&Sinfo,14,NULL
     };
struct Gadget RCa      ={ &RTGad,20,45,60,20,GADGHCOMP,
     ENDGADGET| GADGIMMEDIATE| RELVERIFY,BOOLGADGET| REQGADGET,
     (APTR)&OKRand,NULL,&RCaT,0,0,13,NULL
     };
struct Gadget ROk      ={ &RCa,170,45,60,20,GADGHCOMP,
     ENDGADGET| GADGIMMEDIATE| RELVERIFY,BOOLGADGET| REQGADGET,
     (APTR)&OKRand,NULL,&SaText,0,0,12,NULL
     };
struct Gadget OKGad    ={ NULL,579,235,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&OKText,0,0,11,NULL
     };
struct Gadget SaGad    ={ &OKGad,579,211,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&SaText,0,0,10,NULL
     };
struct Gadget MGad     ={ &SaGad,579,187,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&MText,0,0,9,NULL
     };
struct Gadget RGad     ={ &MGad,579,163,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&RText,0,0,8,NULL
     };
struct Gadget ZGad     ={ &RGad,579,139,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&ZText,0,0,7,NULL
     };
struct Gadget DGad     ={ &ZGad,579,115,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&DText,0,0,6,NULL
     };
struct Gadget SGad     ={ &DGad,579,91,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&SText,0,0,5,NULL
     };
struct Gadget NGad     ={ &SGad,579,67,60,20,GADGHCOMP,
     GADGIMMEDIATE| RELVERIFY,BOOLGADGET,
     (APTR)&OKRand,NULL,&NText,0,0,4,NULL
     };
struct Gadget VGad     ={ &NGad,608,0,30,64,0,0,PROPGADGET,
     (APTR)&VDummy,NULL,NULL,0,(APTR)&VInfo,3,NULL
     };
struct Gadget CGad     ={ &VGad,577,0,30,64,0,0,PROPGADGET,
     (APTR)&CDummy,NULL,NULL,0,(APTR)&CInfo,2,NULL
     };
struct Gadget PGadL    ={ &CGad,546,0,30,256,0,0,PROPGADGET
,
     (APTR)&LDummy,NULL,NULL,0,(APTR)&PLInfo,1,NULL
     };
struct Gadget PGadH    ={ &PGadL,515,0,30,256,0,0,PROPGADGE
T,
     (APTR)&HDummy,NULL,NULL,0,(APTR)&PHInfo,0,NULL
     };
struct Requester Req={ NULL,10,10,250,70,0,0,
     &ROk,&ReqR,&ReqT,0,0,NULL,NULL,NULL,NULL,NULL
     };
struct NewWindow BW    ={
     0,0,640,256,-1,-1,VANILLAKEY| MOUSEBUTTONS| GADGETUP,
     ACTIVATE| BORDERLESS| RMBTRAP,&PGadH,NULL,NULL,NULL,NULL,
     0,0,0,0,WBENCHSCREEN
     };
struct Library            *OpenLibrary();
struct IntuiMessage       *GetMsg();
struct MsgPort            *SPort,*CreatePort();
struct Device             *SDev,*OpenDevice();
struct IntuitionBase      *IntuitionBase;
struct GfxBase            *GfxBase;
struct MathTransBase      *MathTransBase;
struct Window             *win,*OpenWindow();
struct RastPort           *RP;
struct IOAudio            Sound;
VOID closew()                 /* zum Programmende alle Re
sourcen freigeben */
{
   if(!SDev)          CloseDevice(&Sound);
   if(SPort)          DeletePort(SPort,sizeof(struct MsgPor
   t));
   if(Data)           FreeMem(Data,256*sizeof(UBYTE));
   if(win)            CloseWindow(win);
   if(MathTransBase)  CloseLibrary(MathTransBase);
   if(GfxBase)        CloseLibrary(GfxBase);
   if(IntuitionBase)  CloseLibrary(IntuitionBase);
   exit(0);
}
VOID openw()              /* Programmstart: Libraries, Windo
w, ChipMemory ... */
{
   UBYTE *AllocMem();
   if(!(IntuitionBase=(struct IntuitionBase *)
         OpenLibrary("intuition.library",0)))        close
         w();
   if(!(GfxBase=(struct GfxBase *)
         OpenLibrary("graphics.library",0)))         close
         w();
   if(!(MathTransBase=(struct MathTransBase *)
         OpenLibrary("mathtrans.library",0)))        close
         w();
   if(!(win=OpenWindow(&BW)))                        close
   w();
   if(!(Data=AllocMem(256*sizeof(UBYTE),MEMF_CHIP))) close
   w();
   if(!(SPort=CreatePort("SPort",0)))                close
   w();
   Sound.ioa_Request.io_Message.mn_ReplyPort  =SPort;
   Sound.ioa_Request.io_Message.mn_Node.ln_Pri=10;
   Sound.ioa_Data  =&buffer;
   Sound.ioa_Length=(ULONG)sizeof(buffer);
   if(SDev=OpenDevice(AUDIONAME,0,&Sound,0))         close
   w();
   Sound.ioa_Request.io_Command=CMD_WRITE;
   Sound.ioa_Request.io_Flags  =ADIOF_PERVOL| IOF_QUICK;
   RP =win->RPort;
}
ULONG GetMessage()                                  /* Mes
sagePort auslesen */
{
   struct IntuiMessage *mes;
   CLASS=CODE=0;
   ADDR=NULL;
   if(mes=GetMsg(win->UserPort)){           CLASS=mes->Class
   ;
                                            CODE=mes->Code;
                                            ADDR=mes->IAddres
                                            s;
                                            ReplyMsg(mes);
   }
   return(CLASS);
}
VOID SetPoint(x,y)          /* Punkt (incl. Linie) im Zei
chenfeld ausgeben */
   SHORT x;
   UBYTE y;
{
   SHORT col;
   col=2*((x==Ende)||(x==Start));
   Data[x/2]=127-y;
   SetDrMd(RP,JAM1);
   SetAPen(RP,1+col);
   Move(RP,x,0);
   Draw(RP,x,255);
```

```
9k          Move(RP,x+1,0);
Es          Draw(RP,x+1,255);
RG          SetAPen(RP, 2);
xG          WritePixel(RP,x,y);
0Q          WritePixel(RP,x+1,y);
hC0   }
j3    VOID NewW()                /* neue Werte für Period, Cycle
       und Volume lesen */
f8    {
oP4       l=PLInfo.VertPot/GL;
3S        h=PHInfo.VertPot/GH;
0x        p=256*h+1;
jZ        p=(p<124?124:p);
Zi        c=64-CInfo.VertPot/GC;
3Q        c=(c==0?1:c);
RV        v=64-VInfo.VertPot/GV;
rM0   }
4d    Play()
      * Klang abspielen */
pI    {
9M4       NewW();
xL        Sound.ioa_Data  =Data+Start/2;
d7        Sound.ioa_Length=ENDE;
qQ        Sound.ioa_Period=p;
MX        Sound.ioa_Cycles=c;
MX        Sound.ioa_Volume=v;
u4        BeginIO(&Sound);
gL        WaitIO(&Sound);
2X0   }
aS    VOID Save()
OT    {
Xv2     COUNT i,j;
EU      FILE *file;
MZ      NewW();
eQ      if(file=fopen(name,"w")){
tR5        fprintf(file,"/*   -*- %s - erstellt mit Gölzy's SEEd
           -*-   */\n\n",name);
i1         fprintf(file,"#include <devices/audio.h>\n");
ar         fprintf(file,"#include <exec/memory.h>\n");
6z         fprintf(file,"#define Play(); BeginIO(&Sound); WaitIO
           (&Sound);\n\n");
tF         fprintf(file,"UBYTE buffer=15,*SData,SD[%d]={\n\t",END
           E);
Pq         for(i=Start/2,j=0;i<=Ende/2;i++,j++){
RA9            fprintf(file,"%3d",Data[i]);
l1             if(i!=Ende/2)    fprintf(file,",");
SO             if(j==15){       fprintf(file,"\n\t");
uJQ                             j=-1;
Jo9            }
Kp5        }
Rn         fprintf(file,"\n};\n");
dc         fprintf(file,"extern struct MsgPort\t*CreatePort();\n"
           );
9T         fprintf(file,"extern struct Device\t*OpenDevice();\n")
           ;
WZ         fprintf(file,"struct MsgPort\t\t*SPort;\n");
H6         fprintf(file,"struct Device\t\t*SDev;\n");
kr         fprintf(file,"struct IOAudio\t\t Sound;\n\n");
cM         fprintf(file,"VOID SoundOff()\n{\n");
DC         fprintf(file,"\tif(!SDev)\tCloseDevice(&Sound);\n");
Wo         fprintf(file,"\tif(SPort)\tDeletePort(SPort,sizeof(str
           uct MsgPort));\n");
Os         fprintf(file,"\tif(SData)\tFreeMem(SData,%d);\n}\n\n",
           ENDE);
Ml         fprintf(file,"SHORT SoundOn()\n{\n");
t1         fprintf(file,"\tUBYTE *AllocMem();\n");
8G         fprintf(file,"\tCOUNT i;\n\n");
Gd         fprintf(file,"\tif(!(SData=AllocMem(%d,MEMF_CHIP)))\tr
           eturn(1);\n",ENDE);
RI         fprintf(file,"\tfor(i=0;i<%d;i++)\n\t\tSData[i]=SD[i]
           ;\n",ENDE);
y4         fprintf(file,"\tif(!(SPort=CreatePort(\"SPort\",0))){\
           n");
es         fprintf(file,"\t\tSoundOff();\n\t\treturn(1);\n\t}\n")
           ;
Mh         fprintf(file,"\tSound.ioa_Request.io_Message.mn_ReplyP
           ort=SPort;\n");
1K         fprintf(file,"\tSound.ioa_Request.io_Message.mn_Node.l
           n_Pri=10;\n");
fw         fprintf(file,"\tSound.ioa_Data=&buffer;\n");
kz         fprintf(file,"\tSound.ioa_Length=(ULONG)sizeof(buffer)
           ;\n");
zX         fprintf(file,"\tif(SDev=OpenDevice(AUDIONAME,0,&Sound,
           0)){\n");
ky         fprintf(file,"\t\tSoundOff();\n\t\treturn(1);\n\t}\n")
           ;
gC         fprintf(file,"\tSound.ioa_Request.io_Command\t=CMD_WRI
           TE;\n");
it         fprintf(file,"\tSound.ioa_Request.io_Flags\t=ADIOF_PER
           VOL| IOF_QUICK;\n");
gf         fprintf(file,"\tSound.ioa_Data\t\t\t=SData;\n");
gn         fprintf(file,"\tSound.ioa_Length\t\t=%d;\n",ENDE);
kz         fprintf(file,"\tSound.ioa_Period\t\t=(UWORD)%d;\n",p);
3U         fprintf(file,"\tSound.ioa_Volume\t\t=%d;\n",v);
I5         fprintf(file,"\tSound.ioa_Cycles\t\t=(UWORD)%d;\n",c);
F4         fprintf(file,"\treturn(0);\n}\n");
Q7         fclose(file);
rM2     }
jm      else DisplayBeep(0);
t00   }
9F    VOID CheckReq()                                    /* Reques
      ter kontrollieren */
rK    {
Hu4       SHORT reqclear=0;
Uj        while(reqclear==0){
1W8           GetMessage();
4j            if(CLASS==GADGETUP){
61C               switch(GAD_ID){
giG                   case 12:{
JkK                       Save();
Xk                        reqclear=1;
z8                        break;
5aG                   }
or                    case 13:{
boK                       reqclear=1;
3C                        break;
9eG                   }
W0                    default:break;
BgC               }
Ch8           }
Di4       }
EjO   }
Vj    VOID main()
Cf    {
1f4       SHORT x=0,y=0;
p0        BOOL ende=0;
19        COUNT i,j;
mg        FLOAT k;
MO        openw();
5t        SetAPen(RP, 1);
V0        RectFill(RP,0,0,510,256);
Ib        for(i=0;i<512;i+=2)     SetPoint(i,127);
Nk        while(!ende){
0g8           Wait(1<<win->UserPort->mp_SigBit);
Uz            GetMessage();
3h            switch(CLASS){
G7C               case MOUSEBUTTONS:{
q0G                   if(CODE==MENUDOWN){
T1K                       Play();
iq                        while(CODE!=MENUUP) GetMessage();
X2G                   }
Ai                    else if(CODE==SELECTDOWN){
fGK                       while(CODE!=SELECTUP){
d80                           GetMessage();
Om                            if((x!=MOUSX)||(y!=MOUSY)){
mtS                               x=MOUSX;
s1                                y=MOUSY;
xx                                if(x<=510) SetPoint(2*(UBYTE)(
                                  (FLOAT)x/2.0),y);
fA0                           }
gBK                       }
hCG                   }
dm                    break;
jEC               }
7h                case GADGETUP:{
rTH                   switch(GAD_ID){
EOL                       case 4:{     /* Null */
hOP                           for(i=0;i<512;i+=2)     SetPoint(i
                              ,127);
js                            break;
pKL                       }
C5                        case 5:{     /* Sinus */
T6P                           for(j=0;j<2;j++){
iJT                               for(i=0;i<256;i+=2){
```

Listing 1. (Fortsetzung)

```
                        k=sqrt((FLOAT)  ((256-i)*(
                        256-i)+i*i));
                        k=j?-k+128.0:k-128.0;
                        SetPoint(i+256*j,(UBYTE)k)
                        ;
                    }
                }
                break;
            }
            case 6:{    /* 3Eck */
                for(j=0;j<2;j++){
                    for(i=j*256;i<128+j*256;i+=2)
                    {
                        k=(FLOAT)i*.7;
                        k=j?k+204.0:128.0-k;
                        SetPoint(i,(UBYTE)k);
                    }
                    for(i=128+j*256;i<256+j*256;i
                    +=2){
                        k=(FLOAT)i*.7;
                        k=j?-k-28:k-52.0;
                        SetPoint(i,(UBYTE)k);
                    }
                }
                break;
            }
            case 7:{    /* SZahn */
                for(j=0;j<2;j++){
                    for(i=j*256;i<128+j*256;i+=2)
                    {
                        k=.7*(FLOAT)(j*256+180-i);
                        SetPoint(i,(UBYTE)k);
                    }
                    for(i=128+j*256;i<256+j*256;i
                    +=2){
                        k=.7*(FLOAT)(j*256+436-i);
                        SetPoint(i,(UBYTE)k);
                    }
                }
                break;
            }
            case 8:{    /* Recheck */
                for(j=0;j<2;j++){
                    for(i=j*256;i<128+j*256;i+=2)
                        SetPoint(i,38);
                    for(i=128+j*256;i<256+j*256;i
                    +=2) SetPoint(i,218);
                }
                break;
            }
            case 9:{    /* 2mal */
                for(i=j=0;i<256;i+=2,j++) Data[j]
                =Data[i];
                for(i=128;i<256;i++)      Data[i]
                =Data[i-128];
                for(i=0;i<512;i+=2) SetPoint(i,(B
                YTE)(127-Data[i/2]));
                break;
            }
            case 10:{    /* Save */
                Request(&Req,win);
                CheckReq();
                break;
            }
            case 11:{
                ende=1;
                break;
            }
            default:break;
        }
        break;
    }
    case VANILLAKEY:{
        switch(CODE){
            case 49:{    /* Ende links */
                if((Ende>2)&&((Ende-2)>Start)){
                    Ende-=2;
                    SetPoint(Ende+2,(BYTE)(127-Dat
                    a[(Ende+2)/2]));
                    SetPoint(Ende,(BYTE)(127-Data[
                    Ende/2]));
                }
                break;
            }
            case 51:{    /* Ende rechts */
                if(Ende<510){
                    Ende+=2;
                    SetPoint(Ende-2,(BYTE)(127-Dat
                    a[(Ende-2)/2]));
                    SetPoint(Ende,(BYTE)(127-Data[
                    Ende/2]));
                }
                break;
            }
            case 52:{    /* Ende links schnell */
                if((Ende>8)&&((Ende-8)>Start)){
                    Ende-=8;
                    SetPoint(Ende+8,(BYTE)(127-Dat
                    a[(Ende+8)/2]));
                    SetPoint(Ende,(BYTE)(127-Data[
                    Ende/2]));
                }
                break;
            }
            case 54:{    /* Ende rechts schnell */
                if(Ende<506){
                    Ende+=8;
                    SetPoint(Ende-8,(BYTE)(127-Dat
                    a[(Ende-8)/2]));
                    SetPoint(Ende,(BYTE)(127-Data[
                    Ende/2]));
                }
                break;
            }
            case 50:
            case 53:{    /* Ende-Position */
                i=Ende;
                Ende=510;
                SetPoint(i,(BYTE)(127-Data[i/2]));
                SetPoint(510,(BYTE)(127-Data[255])
                );
                break;
            }
            case 55:{    /* Start links */
                if(Start>=8){
                    Start-=8;
                    SetPoint(Start+8,(BYTE)(127-Da
                    ta[(Start+8)/2]));
                    SetPoint(Start,(BYTE)(127-Data
                    [Start/2]));
                }
                break;
            }
            case 57:{    /* Start rechts */
                if((Start<504)&&((Start+8)<Ende)
                ){
                    Start+=8;
                    SetPoint(Start-8,(BYTE)(127-Da
                    ta[(Start-8)/2]));
                    SetPoint(Start,(BYTE)(127-Data
                    [Start/2]));
                }
                break;
            }
            case 56:{    /* Start-Position */
                i=Start;
                Start=0;
                SetPoint(i,(BYTE)(127-Data[i/2]));
                SetPoint(0,(BYTE)(127-Data[0]));
                break;
            }
            default:break;
        }
        break;
    }
    default:break;
  }
 }
 closew();
}
```


Listing 1. (Schluß)

```
#include <devices/audio.h>  /* notwendige Headerfiles einbinden */
#include <exec/memory.h>
#define Play(); BeginIO(&Sound); WaitIO(&Sound);               /*
  Abspielen des Sounds */

UBYTE buffer=15,*SData,SD[56]=
{                        /* globale Variablen und Sounddaten (hier 56)*/
  1,  7, 13, 18, 24, 29, 35, 41, 46, 52, 57, 63, 69, 74, 80, 85,
168,173,179,185,190,196,201,207,213,218,224,229,235,241,246,252,
  1,  7, 13, 18, 24, 29, 35, 41, 46, 52, 57, 63, 69, 74, 80, 85,
168,173,179,185,190,196,201,207
};
extern struct MsgPort   *CreatePort();
extern struct Device    *OpenDevice();
struct MsgPort          *SPort;
struct Device           *SDev;
struct IOAudio           Sound;

VOID SoundOff()                           /* Resourcen freigeben */
{
        if(!SDev)    CloseDevice(&Sound);    /* Sounddevice schließen */
        if(SPort)    DeletePort(SPort,sizeof(struct MsgPort));
                     /* Port */
        if(SData)    FreeMem(SData,56);  /* Speicherbereich
                     freigeben */
}

SHORT SoundOn()       /* Klang initialisieren; Rückgabewert im
                                         Fehlerfalle: 1 */
{
        UBYTE *AllocMem();
        COUNT i;

/* Mit AllocMem() Speicher für Daten im Chipmemory anfordern: */

        if(!(SData=AllocMem(56,MEMF_CHIP)))       return(1);
```

```
        for(i=0;i<56;i++)
                SData[i]=SD[i]; /* Daten ins Chipmemory kopieren */
        if(!(SPort=CreatePort("SPort",0)))
        {                               /* Antwortport kreieren */
                SoundOff();
                return(1);
        }
        Sound.ioa_Request.io_Message.mn_ReplyPort=SPort;
                                                /* Portzuweisung */
        Sound.ioa_Request.io_Message.mn_Node.ln_Pri=10;
                                          /* Prozessespriorität*/
        Sound.ioa_Data=&buffer; /* Soundkanäle festlegen */
        Sound.ioa_Length=(ULONG)sizeof(buffer);
                                          /* Größe des Buffers */
        if(SDev=OpenDevice(AUDIONAME,0,&Sound,0))
        {
                                          /* Audio-Device öffnen */
        SoundOff();                  /* hat nicht geklappt */
        return(1);                   /* daher zurück mit 1 */
        }
        Sound.ioa_Request.io_Command      =CMD_WRITE;
                                          /* Ausgabe vorbereiten */
        Sound.ioa_Request.io_Flags        =ADIOF_PERVOL|IOF_QUICK;
                                          /* Flags */
        Sound.ioa_Data                    =SData;
                                    /* Zeiger auf die Sounddaten */
        Sound.ioa_Length                  =56;
                                    /* Anzahl der Werte (UBYTE) */
        Sound.ioa_Period              =(UWORD)255;  /* Samplerperiode */
        Sound.ioa_Volume                  =64;          /* Lautstärke */
        Sound.ioa_Cycles              =(UWORD)1;   /* Wiederholungen */
        return(0);                              /* Null: alles OK */
}
```

Listing 2. Ein dokumentiertes Listing zum Quellcode einer Sounddatei, die vom SEEd erzeugt wurde



Hatten Sie Probleme mit Artikeln aus dem dritten Sonderheft? Hier finden Sie ergänzende Hinweise und Fehlerkorrekturen.

Guru-Meditation

Sonderheft 3, Seite 28, Amiga-Basic im Höhenflug
Im Listing 37 hat der Fehlerteufel zugeschlagen: In einigen Zeilen waren statt des senkrechten Striches, der sich auf der Amiga-Tastatur rechts oben (neben <←>) befindet, eine »eckige Klammer zu« abgedruckt. Ersetzen Sie einfach die entsprechenden Zeichen, dann läuft das Programm.

Sonderheft 3, Seite 75, Börsenfieber am Amiga
Im Listing 1 »Broker« ist in den Zeilen 557 bis 573 jeweils das Zeichen »eckige Klammer zu« durch den senkrechten Strich zu ersetzen.

Programmservice-Diskette zum Sonderheft 3
Wir hatten leider ein großes Problem mit der Service-Diskette: Diese ist randvoll mit Programmen. »XRef« und die Routinen zum Basic-Kurs waren nicht unterzubringen. Wir haben uns deshalb entschlossen, diese Programme zu allen Programmen aus Sonderheft 4 auf die Service-Diskette zu diesem Heft zu kopieren. Selbstverständlich kopieren wir die Programme auch auf eine leere, formatierte Diskette, wenn Sie uns diese mit einem entsprechenden Hinweis und der Angabe Ihrer Kundennummer einsenden.

Sonderheft 3, Seite 120, Gibber – tödliche Linien
Sollten Sie Schwierigkeiten mit dem Basic-Lader für »Gibber« haben: Laden Sie Amiga-Basic und geben Sie ein:

```
CLEAR,40000
```

Laden Sie den Basic-Lader und verändern Sie die Zeile 3:

```
OPEN "df0:gibber" FOR OUTPUT AS 1
```

Anschließend starten Sie das Basic-Programm. Das lauffähige »Gibber« wird auf der Diskette im internen Laufwerk erzeugt.

Sonderheft 3, Seite 125, Die Rätselmaschine
In das Listing »Textsalat« hat unsere Satzmaschine ein paar falsche Zeichen eingeschmuggelt. In den Zeilen 21, 111 und 134 ersetzen Sie bitte die Zeichen »**2« durch den Ausdruck »^2«. Dann gibt es keinen Dimensionierfehler mehr.

Sonderheft 3, Seite 130, Bodenlos
Im Listing 1 sind in den Zeilen 69 bis 71 die Zeichenfolgen »**« durch das Hochdach »^« zu ersetzen. Die »Klammer auf« in den Zeilen bleibt jeweils stehen.

Sonderheft 3, Seite 111, Moderne Zeiten
Ein ergänzender Hinweis: Die Level-Dateien müssen unbedingt im ASCII-Format gespeichert werden. Der Checkie42 erledigt das automatisch. Vielleicht laden Sie die Datei aber irgendwann mit Amiga-Basic, zum Beispiel um sie zu editieren. Denken Sie bitte daran, das Programm mit dem Kommando »Save"Level",a« zu speichern. Andernfalls speichert Amiga-Basic in einer speziellen Form, und das Hauptprogramm kann die Datei nicht laden.

Sonderheft 3, Seite 159, Checksummer
In Zeile 122 muß die Prüfsumme korrekt lauten: »NK«.

Sonderheft 3, Seite 18, Basic-Dialekte
Auf Seite 26 wird behauptet, GFA-Basic-Programme seien aus rechtlichen Gründen nicht weiterzugeben. Das ist nicht richtig. GFA liefert zusammen mit GFA-Basic den Run-Only-Interpreter GFABASRO aus. Dieser darf mit einem Programm weitergegeben werden. Diese Information ist allerdings nicht in der Version des Handbuchs enthalten, die wir von GFA erhielten. Der einen Satz lange Hinweis versteckt sich in einem zusätzlichen Read-Me-File, in dem auch die Animation von Objekten ausführlich erklärt wird.

und läuft und läuft und läuft und

Laufschriften sind eine Methode, Informationen auszugeben. Sollten Sie vor deren Programmierung zurückschrecken, dann brauchen Sie dieses Utility. Mit »hscroll« bringen Sie Schwung in die Langeweile des CLI.

Schluß mit den Befehlen »Echo« und »Type«, es lebe die Laufschrift! Dabei haben Sie vollen Einfluß auf den scrollenden Text. Dieser kann beliebig plaziert, sowie in unterschiedlicher Breite und mit variabler Geschwindigkeit dargestellt werden.

Unser Tool »hscroll« (Listing 1) ist in Aztec-C geschrieben. Geben Sie das Listing bitte mit dem Checksummer ein. Zur Compilation tippen Sie einfach »cc +l hscroll.c« gefolgt von »ln hscroll.o -lc32« ein. Danach wird es am besten in den C-Ordner kopiert, so daß es jederzeit zur Verfügung steht. Folgende Optionen stehen zur Verfügung, die alle mit einem Leerzeichen getrennt hinter »hscroll« eingegeben werden:

Parameter für hscroll

-lx:	x ist die Position auf der x-Achse für die Laufschrift (Werte von 0 bis 639).
-ty:	y gibt die y-Position an (Werte von 1 bis 256 beim PAL-Amiga).
-wn:	setzt die Breite des Ausgabefensters auf n Pixel. Das Fenster kann maximal 640 Punkte breit sein, wenn Sie Null bei »-l« angeben.
-sn:	n legt die Geschwindigkeit fest, die Werte von 1,2,4 oder 8 annehmen kann.
-rn:	n entspricht hier der Anzahl der Wiederholungen. Bei n=1 läuft Ihr Text nur einmal durch.

Der eigentliche Text der Laufschrift wird mit einem Editor geschrieben und als ASCII-File abgespeichert. Er kann dann ebenfalls als Parameter hinter »hscroll« übergeben werden. Beispiel:

```
hscroll DateiName -l0 -t100 -w200
```

Dieser Aufruf schickt eine Laufschrift am linken Rand des Screens (etwa in der Mitte) über eine Breite von 200 Bildpunkten. Eventuelle Returns innerhalb des Ausgabe-Textes werden durch Leerzeichen ersetzt. Dies ermöglicht ungeahnte Effekte: Lassen Sie Ihre eigenen Fehlermeldungen anzeigen, beim Compilieren den Source-Code durchlaufen oder geben Sie Directories in Laufschrift aus.

Haben Sie eine der Optionen nicht verwendet, so werden Default-Werte benutzt, von denen Sie allerdings nur den für »-r« kennen müssen. Ohne »-r« gibt es eine annähernd unbegrenzte Wiederholungszahl, nämlich genau 2^{32}-1. Dies sind immerhin etwas über 4 Milliarden Wiederholungen; für die meisten Anwendungen dürfte das ausreichen. Dies heißt aber nicht, daß Sie Ihre Laufschrift jetzt eine Woche lang betrachten müssen, sie kann auch mit der Maus weggeklickt werden.

Mehr Effekt mit »Run«

Sie können »hscroll« auch problemlos als Background-Task starten, so daß die Laufschrift im Hintergrund ihrer Wege geht. Dies kann beispielsweise sinnvoll sein, wenn ein längeres Programm geladen werden soll und währenddessen eine Laufschrift das Ganze ein wenig schmücken könnte. Installieren Sie diese also zuerst mit »run hscroll« und den entspechenden Parametern (am besten in einer Batch-Datei) und erst danach kommt der eigentliche Befehl zum Laden. Dabei kann es vorkommen, daß der Ladevorgang geringfügig langsamer wird (je nach Größe der Ausgabe). Soll die Laufschrift noch kurz vor Beendigung des Ladens verschwinden, können Sie die Option »-r« verwenden.

Es gibt fünf Dinge, die Sie bei der Benutzung dieses Tools wissen sollten:

– Das Scrolling ist nicht unbedingt immer stufenlos, wenn die Laufschrift ganz oben im Bildschirm geöffnet wird. Das hängt damit zusammen, daß der Task immer auf die vertikale Austastlücke wartet. Dies ist – vereinfacht gesagt – der Moment, in dem der Elektronenstrahl sich in Zeile 0 befindet. Je weiter oben sich nun die Laufschrift befindet, desto weniger Zeit verbleibt, den Window-»RastPort« zu scrollen, ohne daß zugleich das Bild dargestellt wird. Wenn hierfür nicht genügend Zeit ist, flimmert die Textdarstellung. Der Einfachheit halber wurde auch auf das sogenannte »Double Buffering« verzichtet.

– Sind mit »run hscroll« mehrere Tasks aufgerufen worden, so kann es leicht passieren, daß Ihr CLI Task sozusagen »out of CPU« gerät, also keine Rechenzeit mehr bekommt. Das liegt daran, daß die Laufschrift-Tasks bei Aufruf die höchstmögliche Priorität bekommen, um ein Ruckeln zu verhindern. Man mache sich klar, daß die vertikale Austastlücke etwa 50 mal pro Sekunde auftritt und »hscroll« ebensooft den Prozessor benötigt.

– Es können nur 8 x 8-Punkt Zeichensätze verwendet werden. Wer also in seinem CLI andere benutzen möchte, der sollte erst einmal diese Einschränkung überprüfen. Aus diesem Grunde beträgt die Höhe des geöffneten Scroll-Fensters auch immer 8 Pixel.

– Laufschriften sind nichts anderes als kleine Fenster. Sie können von anderen Fenstern überlappt werden und dann einfach verschwinden.

– Jedes Programm, das längere Blitter-Operationen veranlaßt, ist ein potentieller Störfaktor bezüglich des Filescrollings. Schließlich benutzt auch »hscroll« den Blitter zum Scrolling. Irgendwann aber darf jeder Task mit auch noch so kleiner Priorität mal den Blitter verwenden, was ein leichtes Rukkeln zur Folge hat. Dies kann nicht umgangen werden, wenn man sich konsequent an die Multitasking-Konventionen halten will.

Zur Beschleunigung der Textausgabe gibt es auf dem Amiga einige nützliche Utilities, unter anderem auch »BlitzFonts!« und »FastFonts«, das sich auf der neuen Workbench 1.3 befindet. Diese wirken sich auch günstig auf unsere laufenden Texte aus. Sie sind also unbedingt zu empfehlen, vor allem wegen der geringeren Rechenzeit.

Um dem Laufschrift-Task ein möglichst ruckelfreies Dasein zu bescheren, wird mittels »SetTaskPri(FindTask(0),127);« die Priorität desselben auf den größtmöglichen Wert gesetzt. »FindTask(0)« durchsucht die Liste der gerade ablaufenden Tasks und übergibt den entsprechenden Zeiger auf die TaskStruktur. Nach Beendigung der Hauptschleife wird die Priorität dann auf einen normalen Wert zurückgesetzt. Es werden die üblichen Libraries geöffnet, die für die Verwaltung von Fenstern und Grafik notwendig sind. Um den einzulesenen ScrollText dynamisch verwalten zu können, muß dessen Länge bekannt sein.

Besser geht's mit BlitzFonts!

Es wird daher Speicher für die Daten des »FileInfoBlocks« reserviert, aus dem man nach Untersuchung des Files die Länge desselben entnehmen kann (»FileInfoBlock->fib_Size«).

und läuft und läuft und läuft und

Sozusagen als Unterlage für die Laufschrift wird ein einfaches Fenster »ohne alles« geöffnet, welches im übrigen genau ein Gadget zum Wegklicken enthält.

Die Hauptschleife wartet zunächst mit »WaitTOF();« auf die nächste Austastlücke und scrollt dann den gesamten »RastPort« des Fensters um p Pixels (je nach eingestellter Geschwindigkeit) nach links. Mit Hilfe der Text-Funktion wird dann der neu hinzugekommene Teil neu geschrieben. Das ist in diesem Fall äußerst einfach, weil Operationen im RastPort grundsätzlich nicht über den Rand hinaus gehen, also nichts außerhalb unseres Fensters verändern. Eventuelle Reste werden einfach abgeschnitten.

Smooth mit WaitTOF()

Die Hauptschleife wird solange durchlaufen, bis entweder die Variable für die Wiederholungen abgelaufen ist oder eine Nachricht am Port des Fensters anliegt. Auch dieser Fall ist leicht zu handhaben, da wir nur ein Gadget zur Verfügung haben, das den IDCMP des Fensters ansprechen kann; damit können wir uns eine Auswertung der Nachricht sparen.

Möglicherweise stört Sie die Beschränkung von »hscroll« auf eine Zeile? Die Lösung heißt »vscroll« (Listing 2). Dieses Programm ist in den wesentlichen Punkten genauso aufgebaut. Das Textfile wird jetzt allerdings nach der längsten enthaltenen Zeile abgesucht. Diese bestimmt die Breite des Ausgabefensters. Falls der rechte Rand des Screens überschritten wird, bricht das Programm vorher ab. Selbstverständlich arbeitet »vscroll« auch mit exotischen Formaten wie der 704 Pixel breiten Workbench oder dem PD-Programm »VScreen-Handler« zusammen.

Bitte beachten Sie den Listingausschnitt von Zeile 163 bis Zeile 166. Hier wird dem Betriebssystem suggeriert, es würde nur eine Bitplane bestehen. Durch diesen kleinen Trick erreichen wir eine geringere Belastung des Blitters für die Scroll-Operationen. Vor Beenden des Programms werden die Bitplanes wieder auf Normalwerte gesetzt.

»vscroll« können Sie mit dem Aztec (»cc vscroll.c« gefolgt von »ln vscroll.o -lc«) oder mit dem Lattice (»lc -w vscroll«) compilieren.

Vielleicht fallen Ihnen selbst noch einige Verbesserungen ein, wie zum Beispiel verschiedene Schriftarten, per Tastendruck verschiebbares Fenster oder ähnliches? Wir wünschen Ihnen auf jeden Fall viel Spaß beim Experimentieren mit diesem Utility. *(Christian Wolf/so)*

Programmname:	hscroll
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	Aztek-C
Compiler:	Version3.6
Aufrufe:	cc +l hscroll.c ln hscroll.o -lc32

```
Programmautor: Christian Wolf
-----------------------------

 1 Of0 #include exec/types.h
 2 xV  #include intuition/intuition.h
 3 Dt  #include intuition/intuitionbase.h
 4 d1  #include exec/exec.h
 5 tP  #include exec/execbase.h
 6 Ud  #include libraries/dos.h
 7 v5  #include libraries/dosextens.h
 8 kJ  #include graphics/gfxmacros.h
 9 Ho  #include functions.h
10 Rm  #define NOBORDER NULL
11 YV  #define NOTEXT NULL
12 Fc  /* Gadget zum Abklicken... */
13 EM  struct Gadget ClickGadget = { NULL,0,0,0,8,GADGHNONE,RELVER
       IFY,
14 5vU                              BOOLGADGET,NOBORDER,0,NOTEXT,
                                    0,0,NULL,0};
15 t90 /* Einfaches Fenster ohne alles... */
16 99  struct NewWindow NewScrollWindow = {
17 Ch7         0,230,640,8,0,1,
18 V9          GADGETUP,
19 x6          BORDERLESS,
20 ZJ          &ClickGadget,NULL,
21 wB          NULL,
22 Y2          NULL,NULL,
23 xT          0,0,0,0,
24 SW          WBENCHSCREEN };
25 Ls0 /* Das Übliche... */
26 tM  struct IntuitionBase *IntuitionBase;
27 Dq  struct GfxBase *GfxBase;
28 wT  /* Default scrolltext */
29 Ag  char *DefaultText= "Hscroll 1.0 (w) 1989 by Christian Wolf.
        Options: -lx and -ty"
30 RtJ                    " puts your scroll to equivalent x and y
                           position. Use -w to"
31 TJ                     " select the window's width. -s is used
                          to have different"
32 iP                     " scroll speeds. Values allowed are: 1,2
                          ,4,8. Add a filename"
33 h0                     " to your parameter list to have your ow
                          n text. -rn lets"
34 bQ                     " the text repeat n times. To remove the
                           scrolltext, just"
35 ft                     " click it. Have much fun...
                                          ";
36 e00 main(argc,argv)
37 1b1  int argc;
38 dB   char **argv;
39 Z20 {
40 Pr1  struct Window *ScrollWindow;
41 qQ   struct RastPort *ScrollRastPort;
42 Wd   short l=160,t=248,w=320,s=2,p=0,c=0,sl=strlen(DefaultText)
        ;
43 qG   long r=0xffffffff,Size;
44 Dh   struct FileHandle *ThisFile;
45 FN   struct FileLock *MyLock;
46 6F   char *ScrollText=DefaultText,*MyPtr;
47 iD   struct FileInfoBlock *FileInfoBlock;
48 AY   /* Damit es nicht so ruckelt -> höchste Taskprorität */
49 CR   SetTaskPri(FindTask(0),127);
50 aA   /* FileInfoBlock öffen... */
51 yK   FileInfoBlock=AllocMem(sizeof(struct FileInfoBlock),MEMF_P
        UBLIC|MEMF_CLEAR);
52 Pf   if (!FileInfoBlock) exit(0);
53 Oy   /* parameter liste durchgehen */
54 uJ   argc--;
55 CO   argv++;
56 8D   while (argc--)
57 rK8          {
58 um9           if (**argv=='-')
59 tMD               {
60 o2E                (*argv)++;
61 an                 switch((*(*argv)++))
62 wPL                        {
63 ErM                         case 'w':
64 d9                          w=atoi(*argv);
65 y7                          break;
66 kC                          case 'l':
67 Kf                          l=atoi(*argv);
68 1A                          break;
69 Bl                          case 't':
70 d6                          t=atoi(*argv);
71 4D                          break;
72 Bk                          case 's':
73 e6                          s=atoi(*argv);
74 7G                          break;
75 Bj                          case 'r':
76 f6                          r=atoi(*argv);
77 AJ                          break;
78 G1L                        }
79 HmD               }
```

Listing 1. »hscroll.c« zaubert Ihnen eine horizontale Laufschrift auf den Bildschirm

```
80 7u9         else
81 FiD             {
82 YKE             /* TextFile laden... */
83 Wj              if (ThisFile=Open(*argv,MODE_OLDFILE))
84 I1I                 {
85 qNJ                 MyLock=Lock(*argv,ACCESS_READ);
86 xI                  Examine(MyLock,FileInfoBlock);
87 2d                  Size=FileInfoBlock->fib_Size;
88 iW                  if (MyPtr=ScrollText=AllocMem(Size,MEMF_
                       PUBLIC))
89 NqO                     {
90 7WP                     Read(ThisFile,ScrollText,Size);
91 dj                      MyPtr[Size]=0;
92 QR                      /* eventuelle returns entfernen...
                            */
93 Db                      while (*MyPtr)
94 SvW                         {
95 2WX                         if (*MyPtr == 0x0a) *MyPtr
                               ++=' ';
96 Z9                          else MyPtr++;
97 Z4W                         }
98 sHP                     UnLock(MyLock);
99 B4                      Close(ThisFile);
100 c70                    }
101 QdJ                else ScrollText=DefaultText;
102 e9I                }
103 hPE            sl=strlen(ScrollText);
104 gBD            }
105 Oo9        argv++;
106 iD8        }
107 Q51 /* Libries öffnen... */
108 DE  IntuitionBase=(struct IntuitionBase *)OpenLibrary("intuiti
    on.library",0);
109 mQ  GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0
    );
110 Tr  /* Werte für Fenster und Gadget initialisieren */
111 xu  NewScrollWindow.LeftEdge=l;
112 U1  NewScrollWindow.TopEdge=t;
113 wp  NewScrollWindow.Width=w;
114 Wz  ClickGadget.LeftEdge=0;
115 LK  ClickGadget.TopEdge=0;
116 Lv  ClickGadget.Width=w;
117 Bh  /* Fenster öffnen */
118 Sk  if (!(ScrollWindow=OpenWindow(&NewScrollWindow))) exit(0);
119 gF  /* Rasterport holen */
120 To  ScrollRastPort=ScrollWindow->RPort;
121 vv  SetAPen(ScrollRastPort,3);
122 mk  SetBPen(ScrollRastPort,0);
123 Zq  SetDrMd(ScrollRastPort,JAM1);
124 dx  /* Hauptschleife -> Scrollen, solange r!=0 oder UserPort
    leer */
125 UX  while (!GetMsg(ScrollWindow->UserPort) && r)
126 yR2  {
127 5J3   /* Auf den VBL warten */
128 ij    WaitTOF();
129 MR    /* Grafik um s Pixels verschieben... */
130 Sv    ScrollRaster(ScrollRastPort,s,0,0,0,w-1,7);
131 S7    Move(ScrollRastPort,w-s-p,6);
132 kC    if (sl <= c)
133 5Y7       {
134 wB8       c=0;
135 Te        r--;
136 Ch7       }
137 eV3   /* Nachfolgenden Text reinschreiben */
138 MW    Text(ScrollRastPort,&ScrollText[c],1);
139 X4    p+=s;
140 9w    if (p > 8-s)
141 Dg7       {
142 Uw8       p=0;
143 tl        c++;
144 Kp7       }
145 Lq2  }
146 pil /* Alles schließen... */
147 ER  CloseWindow(ScrollWindow);
148 fE  CloseLibrary(IntuitionBase);
149 pL  CloseLibrary(GfxBase);
150 W8  if (ScrollText!=DefaultText) FreeMem(ScrollText,sl);
151 s9  if (FileInfoBlock) FreeMem(FileInfoBlock,sizeof(struct Fil
    eInfoBlock));
152 r9  SetTaskPri(FindTask(0),0);
153 TyO }
```


Listing 1. (Schluß)

Programmname:	**vscroll**
Computer:	**A500, A1000, A2000 mit Kickstart 1.2**
Sprache:	**C**
Compiler:	**Aztec oder Lattice**
Aufrufe:	**siehe Text**

```
Programmautor: Christian Wolf
-----------------------------

1 I60 #include <intuition/intuitionbase.h>
2 On  #include <libraries/dos.h>
3 AF  #include <exec/memory.h>
4 bk  #include <graphics/gfxbase.h>
5 51  #ifdef LATTICE
6 Zd  #include <proto/all.h>
7 p6  #else
8 iM  #include <functions.h>
9 QM  #endif
10 Aw #define fib FileInfoBlock
11 Cd #define fibsize (long)sizeof(struct fib)
12 To #define NOBORDER NULL
13 aX #define NOTEXT NULL
14 He /* Gadget zum Abklicken... */
15 ss struct Gadget ClickGadget = { NULL,0,0,0,0,GADGHNONE,RELVER
   IFY,
16 7xU                             BOOLGADGET,NOBORDER,0,NOTEXT,
                                   0,0,NULL,0};
17 vBO /* Einfaches Fenster ohne alles... */
18 BB struct NewWindow NewScrollWindow = {
19 Ej7        0,230,640,8,0,1,
20 u8         GADGETUP|GADGETUP,
21 z8         BORDERLESS,
22 bL         &ClickGadget,NULL,
23 yD         NULL,
24 a4         NULL,NULL,
25 5z         640,200,640,200,
26 UY         WBENCHSCREEN };
27 NuO /* Das Übliche... */
28 vO struct IntuitionBase *IntuitionBase;
29 Fs struct GfxBase *GfxBase;
30 8t struct Table { char *Line;
31 oAF             long Len; };
32 zfO struct Table DefaultText[] = { "           VScroll 1.0
            ",37,
33 2ZV                               "    (c) 1989 by Markt und Te
                                   chnik    ",37,
34 Jn                                "     written by Christian W
                                   olf     ",37,
35 YA                                "      currently resident at
                                          ",37,
36 nm                                "       Spandauer Str. 104L
                                          ",37,
37 PT                                "         1 Berlin 20
                                          ",37,
38 vk                                "       Tel.: 030/3660450
                                          ",37,
39 I7                                " Usage: vscroll textfile [
                                   options]  ",37,
40 1t                                "  Use -w to modify window's
                                    width    ",37,
41 G7                                "  Use -s number of jiffies
                                   1,2,4,8   ",37,
42 nV                                "  Use -l and -t to position
                                    window   ",37,
43 8E                                "  Use -r for no. of repetit
                                   ions      ",37,
44 Vu                                "  Use -h for a new height o
                                   f window  ",37,
45 cm                                "
                                          ",37,
46 dn                                "
                                          ",37,
47 eo                                "
                                          ",37,
48 Gc                                "
                                          ",37,0 };
49 LBO struct BitMap ScrollBitMap;
50 fx void main(argc,argv)
51 q01  char **argv;
52 Gq   int argc;
53 nGO {
```

```
struct Table *ScrollText=DefaultText,*cl=DefaultText;
char *MyPtr,*LoadText=0;
struct Window *ScrollWindow;
struct RastPort *ScrollRastPort;
long l=160,t=60,w=37*8,s=1,p=0,c=0,h=100,sl;
long r=0xffffffff,Size,Lines;
struct fib *fib;
struct FileHandle *ThisFile;
struct FileLock *MyLock;
int i;
/* Damit es nicht so ruckelt -> höchste Taskprorität */
SetTaskPri(FindTask(0L),127L);
/* parameter liste durchgehen */
argc--;
argv++;
while (argc--)
    {
    if (**argv=='-')
        {
        (*argv)++;
        switch((*(*argv)++))
            {
            case 'h':
            h=atoi(*argv);
            break;
            case 'l':
            l=atoi(*argv);
            break;
            case 't':
            t=atoi(*argv);
            break;
            case 's':
            s=atoi(*argv);
            break;
            case 'r':
            r=atoi(*argv);
            break;
            }
        }
    else
        {
        if (ScrollText==DefaultText)
            {
            /* TextFile laden... */
            if (ThisFile=(struct FileHandle *)Open(*argv,MODE_OLDFILE))
                {
                /* fib öffen... */
                if (fib=(struct fib *)AllocMem(fibsize,MEMF_PUBLIC))
                    {
                    MyLock=(struct FileLock *)Lock(*argv,ACCESS_READ);
                    Examine(MyLock,fib);
                    UnLock(MyLock);
                    Size=fib->fib_Size;
                    if (LoadText=(char *)AllocMem(Size,MEMF_PUBLIC))
                        {
                        Read(ThisFile,LoadText,Size);
                        LoadText[Size-1]=0;
                        for (MyPtr=LoadText,Lines=2;*MyPtr;MyPtr++)
                            {
                            if (*MyPtr == 0x0a) Lines++;
                            }
                        if (!(cl=ScrollText=(struct Table *)AllocMem(Lines<<3,MEMF_PUBLIC|MEMF_CLEAR))) exit(0);
                        for (MyPtr=LoadText,cl++->Line=LoadText;*MyPtr;)
                            {
                            if (*MyPtr == 0x0a)
                                {
                                *MyPtr=0;
                                cl++->Line=++MyPtr;
                                }
                            else MyPtr++;
                            }
                        cl->Line=0;
                        for (cl=ScrollText,w=0;cl->Line;cl++)
                            {
                            if ((cl->Len=strlen(cl->Line)) > w) w=cl->Len;
                            }
                        w<<=3;
                        cl=ScrollText;
                        }
                    else ScrollText=DefaultText;
                    FreeMem(fib,fibsize);
                    }
                else ScrollText=DefaultText;
                Close(ThisFile);
                }
            }
        }
    argv++;
    }
/* Libries öffnen... */
IntuitionBase=(struct IntuitionBase *)OpenLibrary("intuition.library",0L);
GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0L);
/* Werte für Fenster und Gadget initialisieren */
NewScrollWindow.LeftEdge=l;
NewScrollWindow.TopEdge=t;
NewScrollWindow.Width=w;
NewScrollWindow.Height=h;
ClickGadget.LeftEdge=0;
ClickGadget.TopEdge=0;
ClickGadget.Height=h;
ClickGadget.Width=w;
/* Fenster öffnen */
if (ScrollWindow=(struct Window *)OpenWindow(&NewScrollWindow))
    {
    /* Rasterport holen */
    ScrollRastPort=ScrollWindow->RPort;
    SetAPen(ScrollRastPort,1L);
    SetBPen(ScrollRastPort,0L);
    SetDrMd(ScrollRastPort,JAM1);
    /* BitMap kopieren */
    ScrollBitMap=*ScrollRastPort->BitMap;
    ScrollRastPort->BitMap=&ScrollBitMap;
    ScrollBitMap.Depth=1;
    h+=6;
    while (!GetMsg(ScrollWindow->UserPort) && r)
        {
        WaitTOF();
        ScrollRaster(ScrollRastPort,0L,s,0L,0L,w-1,h-1);
        Move(ScrollRastPort,0L,h-s-p);
        Text(ScrollRastPort,cl->Line,cl->Len);
        p+=s;
        if (p > 8-s)
            {
            p=0;
            cl++;
            if (!cl->Line)
                {
                cl=ScrollText;
                r--;
                }
            }
        }
    ScrollBitMap.Depth=2;
    /* alles schließen */
    CloseWindow(ScrollWindow);
    }
CloseLibrary((struct Library *)IntuitionBase);
CloseLibrary((struct Library *)GfxBase);
if (ScrollText!=DefaultText) FreeMem(ScrollText,Lines<<3);
if (LoadText) FreeMem(LoadText,Size);
SetTaskPri(FindTask(0L),0L);
}
```


Listing 2. Mit »vscroll« bewegen Sie ein ganzes Textfile vertikal über den Screen

Doppelt

Haben Sie sich schon einmal wünscht, damit Sie sich nicht ständig müssen? Vergessen Sie's. Wir zei

Sie wollen im CLI weiterarbeiten, während Sie den Aufbau einer Grafik beobachten? Sie möchten beim Debuggen nicht ständig die Fenster durchschalten? Oder wollen Sie im Editor die letzten Ausgaben Ihres Programms nicht aus den Augen verlieren? Die Lösung heißt »Dualplayfield-Modus«.

Wenn Sie in Zukunft für Ihre Programme mehr Platz brauchen als ein Screen bieten kann, legen Sie einfach einen Teil der Ausgaben auf einen durchsichtigen Screen. Wenn Sie glauben, die Programmierung wäre zu aufwendig, verwenden Sie die Routinen aus »Dual.c« (Listing 1).

Mit zwei neuen Grafikfunktionen und einigen Erläuterungen zum Sachverhalt schaffen wir allen C-Programmierern die Grundlage zu einer komfortablen Nutzung dieser einmaligen Fähigkeit. Um die Sache erst einmal locker angehen zu lassen, tippen Sie zunächst das Listing ein und compilieren es mit »lc -L dual« (wenn Sie einen Lattice-Compiler haben) oder mit »cc +l dual«, gefolgt von »ln dual -lc32« als Aztec-Besitzer. Mit Eingabe von »dual« starten Sie unser Demo-Programm. Wenn alles geklappt hat, wird die Workbench plötzlich rot übermalt, um dann mit rotem Hintergrund wieder aufzutauchen und schließlich wieder ihre alte Hintergrundfarbe anzunehmen. Dieser Vorgang wiederholt sich zehnmal, bevor das Programm selbständig abbricht.

Wer aufmerksam gewesen ist, der wird bemerkt haben, daß während des Malvorganges der Inhalt der Workbench-Grafik nicht verändert wurde. Damit sind wir auch schon beim entscheidenden Punkt:

Grafiken überlagern sich

Der Dualplayfield-Modus erlaubt die gleichzeitige und voneinander völlig unabhängige Darstellung zweier »Playfields«, ohne daß diese sich in irgendeiner Weise gegenseitig beeinflussen. Dabei spielt lediglich die Priorität der Darstellungsflächen eine Rolle, da das eine Playfield durch das andere überlagert wird. Der Effekt der Überlagerung kommt dadurch zustande, daß die vordere Grafik immer mit transparenter Hintergrundfarbe definiert ist. Dadurch kommt sozusagen statt der Hintergrundfarbe die untere Grafik durch. Bei unserem Demo ist anfangs offensichtlich die Workbench überlagert, beim Löschen die vordere Grafik.

Nachdem die grundlegenden Prinzipien nun etwas vertrauter sind, können Sie sich an die Einbindung unseres Dualplayfield-Tools in Ihre eigenen Programme machen. Am besten compilieren Sie das Listing ohne den »main«-Teil, so daß die Möglichkeit besteht, unsere Funktionen später als Modul hinzuzulinken. Um damit eine Grafik als Vordergrund zu aktivieren, fügen Sie die Funktion »MakeDual(Depth,ScreenTitle)« zu Ihrem C-Quelltext hinzu. Dabei gibt »Depth« die Anzahl der Bitplanes, also die Farbvielfalt und »ScreenTitle« den Namen des Screens an, in dem wir fortan weiterarbeiten wollen, üblicherweise ist das der WorkbenchScreen. Sollte die Funktion einen Wert ungleich Null zurückgeben, dann ist irgendetwas schiefgegangen. Mögliche Fehlerquellen sind zu wenig Speicher, ein nicht gefundener Screen oder aber eine unzulässige Anzahl von Bitplanes – hier sind nur die Werte 1 oder 2 zugelassen. Wurde die Funktion in Ihrem Programm aus Versehen zum zweiten Mal aufgerufen, dann wird dies ebenfalls entsprechend quittiert. Die Installation selbst bringt keine sichtbaren Veränderungen mit sich, und das Vorhandensein wird erst bei nachfolgendem Ansprechen des Bildschirmspeichers bemerkt.

Einfache Installation

Um Grafikoperationen in die Wege zu leiten, werden zwei Strukturen zur Verfügung gestellt, die dem Amiga-Grafik-Programmierer nicht ganz unbekannt sein dürften: »struct RastPort DualRastPort« und »struct BitMap DualBitMap«. Sie wurden bei Aufruf von »MakeDual« initialisiert und gewährleisten die Benutzung aller im Amiga residenten Grafikfunktionen, die fast immer auf diese Strukturen zurückgreifen. Unser Demo zeigt dies ganz deutlich. Nach ein paar Versuchen werden Sie problemlos damit zurechtkommen. Beachten Sie an dieser Stelle bitte noch, daß die neuen Grafik-Tools sowohl die »Graphics-« als auch »Intuition.library« benötigen. Beide werden vom Modul aus geöffnet.

Wie schon erwähnt kann man die Priorität der Playfields selbst bestimmen. Um diese Fähigkeit des Amiga zu nutzen, bedienen Sie sich einfach der Funktion »DualTo(FRONT)«, um die Workbench beziehungsweise den Screen Ihrer Wahl zu überlagern. Die gegenteilige Wirkung erreicht man mit dem Funktionsargument »BACK«, damit wird die Grafik einfach in den Hintergrund gelegt. Unser kurzes Demo demonstriert dies deutlich.

Jeder Programmierer sollte bei seinen Programmen dafür sorgen, daß bei Programmende sämtlicher reservierter Speicher freigegeben und geöffnete Libraries wieder geschlossen werden. Haben Sie irgendwann einmal den Dualplayfield-Modus aktiviert, so bleibt dieser solange erhalten, bis vom Programm die Anweisung »CloseDual();« angetroffen wird. Hiermit sorgen Sie für die korrekte Rückgabe des Speichers an das System, für das Schließen der Libraries und für den normalen Workbench-Betrieb. Schließen Sie Ihren C-Quelltext immer mit dieser Zeile ab, denn nur so hält man sich so an die Konventionen des Multitaskings.

Mit dem Einschalten des Dualplayfield-Modus bietet sich dem Grafikprogrammierer beispielsweise auch eine praktische Hilfe beim Debuggen seiner Programme. Stellen Sie sich folgende Situation vor: Als stolzer Besitzer des Aztec-C-Entwicklungssystems inklusive Source-Level-Debugger sind Sie gerade dabei, ein Programm zu entwerfen, das unter Benutzung eines eigenen Screens als Grafikunterlage Ihre kreativen Ideen in die Tat umsetzt, aber partout nicht laufen will. Dies endet meist mit dem Einsatz des Debuggers, und es wird Zeile für Zeile überprüft. An den (vermutlich) kritischen Stellen schaltet man die Grafik in den Vordergrund und schaut nach, was passiert ist, schaltet zurück und führt die nächste Zeile aus, schaltet vor, schaltet zurück – ein Schalten ohne Ende. Wünschenswert wäre also die Möglichkeit, sowohl Debugger als auch Grafik auf einen Blick zu haben. Nichts leichter als das! Jetzt wird einfach »dual.c« hinzugefügt und »MakeDual(1,"Workbench-Screen");« statt »OpenScreen();« benutzt. Da wir sowohl BitMap- als auch Rastport-Struktur zur Verfügung haben, ist ein problemloses Verlagern der Grafik möglich. Sie sollten sich jetzt nicht beklagen, daß Sie nur noch maximal vier Farben zur Verfügung haben, es geht schließlich nur darum, den Fehler im Algorithmus zu finden.

Wenn zuviel der Fläche des Vordergrund-Playfields für Ihre Grafik verwendet wird, dann sollte man dieses vorher besser mit »DualTo(BACK);« in den Hintergrund schalten. Die übliche CLI- oder SDB-Umgebung läßt meist noch genügend Platz zum »Durchblick« auf die Ausgaben.

Nichts ohne Wermutstropfen

Zum Schluß möchten wir Sie noch auf einen Nachteil des Dualplayfield-Modus hinweisen, den Sie in Erwägung ziehen sollten. Bei zwei sich überlagernden Grafiken, die womöglich noch beide aus zwei Bitplanes bestehen, müssen in gleicher Zeit genau doppelt soviel Daten aus dem Bildspeicher ausgelesen werden, um dann zum Bildschirm zu wandern. Bei »LORES«-Grafik, die von den meisten Programmen für diese Zwecke benutzt wird, hat das noch keine bemerkenswerten Folgen. Bei »HIRES«-Grafik, wie der Workbench, müssen mit 640 Pixel pro horizontaler Zeile die doppelte Anzahl Worte aus dem Speicher geholt werden. Das hat erhebliche Geschwindigkeitsverluste für den Prozessor zur Folge, da die Grafikchips für ein störungsfreies Bild auf dem Monitor Vorrang beim Speicherzugriff haben. Somit muß der Prozessor im Zweifelsfall warten.

Bei dem eben erwähnten Beispiel beträgt der Geschwindigkeitsverlust bis zu 50 Pro-

es Spiel

einen zweiten Monitor ge- durch alle Screens und Fenster klicken gen Ihnen eine einfachere Lösung.

zent. Besitzer einer externen Speichererweiterung sind davon nicht so hart betroffen, denn in diesem Fall liegen die Programme meist im Fast-Memory, das von den Grafikchips nicht erreicht wird. Dazu gleich die Bestätigung: Lassen Sie das Demo-Programm noch einmal laufen, wobei gleichzeitig der WorkbenchScreen heruntergezogen sein sollte. Es läßt sich eindeutig beobachten, daß »dual« um so schneller wird, je weiter der Screen nach unten wandert, also je weniger von ihm dargestellt wird. Das bedeutet letztendlich die Freigabe von Zugriffen auf den Datenbus, weil dieser nicht so häufig von den Grafik-Chips benötigt wird.

Aber auch so dürfte die Leistung Ihres Amiga wohl kaum zu stark beeinträchtigt sein. Wir wünschen Ihnen viel Spaß mit diesem neuen Utility.

(Christian Wolf/so)

Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec oder Lattice
Aufrufe:	siehe Text

```
Programmautor: Christian Wolf
-----------------------------

oI0 /***********************************/
Vu  /* dual.c  1989 by Christian Wolf */
x1  /*    in Spandauer Str. 104L       */
Oi  /*         1 Berlin 20             */
iA  /*     Tel.: 030/3660450           */
dh  /* bugs fixed by Heiko Schlichting */
u0  /***********************************/
PD  #include <intuition/intuitionbase.h>
GL  #include <exec/memory.h>
hq  #include <graphics/gfxbase.h>
Br  #ifdef LATTICE
fj  #include <proto/all.h>
vC  #else
oS  #include <functions.h>
WS  #endif
Jz  #define FRONT 1
76  #define BACK 0
1Y  /* Fehlermeldungen: */
15  #define NOTITLE               1
Sx  #define OUTOFMEM              2
Kt  #define SORRY_SCREENNOTFOUND 3
MM  #define WRONGDEPTH            4
y1  #define ALREADYSTARTED        5
8i  /* Das übliche ... */
sL  struct IntuitionBase *IntuitionBase;
Cp  struct GfxBase *GfxBase;
Ds  /* Für den Benuzter */
XI  struct BitMap *DualBitMap;
ow  struct RastPort *DualRastPort;
ip  /* für private Zwecke, am besten unberührt lassen */
lt  static struct RasInfo *DualRasInfo;
UW  static long planesize;
E6  static struct BitMap *CBitMap;
GX  static struct Screen *CScreen;
dA  static BOOL StartFlag=FALSE;
PF  static struct View *MyView;
rR  /* Playfield nach vorn oder nach hinten */
qe  void DualTo(Front)
AQ1  BOOL Front;
a30 {
sU1  if (Front)
c55      {
la6       CScreen->ViewPort.Modes| =DUALPF| PFBA;
pJ        MyView->Modes| =DUALPF| PFBA;
jE5      }
ZM1  else
hA5      {
sU6       CScreen->ViewPort.Modes| =DUALPF;
g5        CScreen->ViewPort.Modes&= (PFBA^0xffff);
oJ5      }
2x1  /* keine Guru beim Verschieben des Screens */
uH   Forbid();
iM   MakeVPort(MyView,MyView->ViewPort);
dF   /* Um ein Flimmern zu verhindern */
Ta   WaitBOVP(MyView->ViewPort);
I9   MrgCop(MyView);
uD   /* Display auf jeden Fall einschalten */
AI   LoadView(MyView);
G4   Permit();
yT0 }
sq  /* Alles wieder freigeben */
jT  void CloseDual()
xQ  {
Ri1  if (!StartFlag) return;
jD   CScreen->ViewPort.RasInfo->Next=0;
ks   CScreen->ViewPort.Modes^=DUALPF;
ob   MyView->Modes^=DUALPF;
AX   Forbid();
yc   MakeVPort(MyView,MyView->ViewPort);
WN   MrgCop(MyView);
SG   Permit();
xF   if (DualBitMap->Planes[0]) FreeMem(DualBitMap->Planes[0]
     ,planesize);
90   if (DualBitMap->Planes[1]) FreeMem(DualBitMap->Planes[1]
     ,planesize);
n9   if (DualRasInfo) FreeMem(DualRasInfo,(long)sizeof(struct R
     asInfo));
es   if (DualBitMap) FreeMem(DualBitMap,(long)sizeof(struct Bit
     Map));
2H   if (DualRastPort) FreeMem(DualRastPort,(long)sizeof(struct
      RastPort));
JT   CloseLibrary((struct Library*)IntuitionBase);
CD   CloseLibrary((struct Library*)GfxBase);
mn   StartFlag=FALSE;
In0 }
90  MakeDual(Depth,ScreenTitle)
jZ1  short Depth;
60   char *ScreenTitle;
I10 {
iz  /* Argumente checken */
pI1  if (StartFlag) return ALREADYSTARTED;
c1   if (Depth < 1 || Depth > 2) return(WRONGDEPTH);
kB   if (!ScreenTitle) return(NOTITLE);
kT0 /* Intuition- und GfxLibrary öffnen, geht NIE schief */
dM1  IntuitionBase=(struct IntuitionBase *)OpenLibrary("intuiti
     on.library",0L);
RF   GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0
     L);
at   /* Speicher für verschiedene Strukturen reservieren */
mY   if (!(DualRasInfo=AllocMem((long)sizeof(struct RasInfo),ME
     MF_PUBLIC)))
Sv5      {
m16       CloseDual();
4g        return OUTOFMEM;
Z45      }
b81  if (!(DualBitMap=AllocMem((long)sizeof(struct BitMap),MEMF
     _PUBLIC)))
X05      {
r66       CloseDual();
91        return OUTOFMEM;
e95      }
ef1  if (!(DualRastPort=AllocMem((long)sizeof(struct RastPort),
     MEMF_PUBLIC)))
c55      {
wB6       CloseDual();
Eq        return OUTOFMEM;
jE5      }
4g1  /* Screen mit richtigem Namen suchen */
nD   CScreen=IntuitionBase->FirstScreen;
QM   while (CScreen)
jC8        {
om9         /* Name gefunden -> DualPlayField aktivieren */
```

Listing 1. »dual.c« ist ein Leckerbissen für alle Grafik-Freaks

```
        if (!strcmp(ScreenTitle,CScreen->Title))
            {
            CBitMap=&CScreen->BitMap;
            /* BitMap kopieren */
            *DualBitMap=*CBitMap;
            /* RastPort kopieren... */
            *DualRastPort=CScreen->RastPort;
            /* ...und BitMap korrigieren */
            DualRastPort->BitMap=DualBitMap;
            MyView=GfxBase->ActiView;
            planesize=CBitMap->BytesPerRow*CBitMap->Rows;
            if (!(DualBitMap->Planes[0]=
                  AllocMem(planesize,MEMF_CHIP|MEMF_CLEAR
                  )))
                {
                CloseDual();
                return OUTOFMEM;
                }
            if (Depth==2)
                {
                if (!(DualBitMap->Planes[1]=
                      AllocMem(planesize,MEMF_CHIP|MEMF_
                      CLEAR)))
                    {
                    CloseDual();
                    return OUTOFMEM;
                    }
                }
            else DualBitMap->Planes[1]=0;
            DualBitMap->Depth=Depth;
            DualRasInfo->BitMap=DualBitMap;
            DualRasInfo->RxOffset=0;
            DualRasInfo->RyOffset=0;
            DualRasInfo->Next=0;
            CScreen->ViewPort.RasInfo->Next=DualRasInfo;
            DualTo(FRONT);
            StartFlag=TRUE;
            return TRUE;
            }
            CScreen=CScreen->NextScreen;
            }
    return SORRY_SCREENNOTFOUND;
}
/* ein kleines Test-Programm */
void main()
{
    long x,i;
    int Err;
    /* Dualplayfieldmode aktivieren */
    if (!(Err=MakeDual(1,"Workbench Screen")))
        {
        printf("Something has gone wrong...Error Nr. %d\n",Err);
        exit(0);
        }
    /* und ein bißchen malen */
    for (i=0;i<10;i++)
        {
        DualTo(FRONT);
        SetAPen(DualRastPort,1L);
        SetDrMd(DualRastPort,JAM1);
        for (x=0;x<639;x++)
            {
            Move(DualRastPort,x,0L);
            Draw(DualRastPort,x,255L);
            }
        DualTo(BACK);
        SetAPen(DualRastPort,0L);
        for (x=639;x>0;x--)
            {
            Move(DualRastPort,x,0L);
            Draw(DualRastPort,x,255L);
            }
        }
    /* wieder schließen */
    CloseDual();
}
(C) 1988 M&T
```

Listing 1. »dual.c« (Schluß)

Requester ade

Meldungen wie »No disk present in unit 0« oder »Please insert Volume Workbench in any drive« rauben dem Benutzer schon mal den letzten Nerv. Unser neuer CLI-Befehl »remreq« macht mit diesen Requestern endgültig Schluß.

Sie haben Bedenken? Stellen Sie sich folgende Situation vor: Man möchte beim Booten den Verlauf der »startup-sequence« vom Vorhandensein bestimmter Disketten oder Laufwerke abhängig machen. So sollte es beispielsweise möglich sein, bei Vorliegen der Modula-Diskette das System auf Modula vorzubereiten, bei anderen Disketten entsprechend andere Maßnahmen zu ergreifen. Eine Fallunterscheidung würde man mit »IF EXISTS MODULA:« oder »IF EXISTS C-Workdisk:« erreichen. Dies hätte das Erscheinen eines oder mehrerer Requester zur Folge, was nicht gerade praktisch ist, da der Bootvorgang so die Aufmerksamkeit und den Eingriff des Benutzers erfordert.

Mit »remreq« (Listing 1) wird das anders. Geben Sie das Listing ein und compilieren es. Das Programm läuft sowohl mit Aztec- wie auch mit Lattice-C. Mit Lattice compilieren Sie mit »lc -L remreq«, Aztek-Besitzer verwenden »cc remreq« gefolgt von »ln remreq -lc«.

Wenn Sie die Wirkungsweise des Programms testen wollen, geben Sie im CLI »remreq« ein, entnehmen eine eventuell vorhandene Diskette aus Laufwerk df0: und geben dann »dir df0:« ein. Die Folge ist eine Fehlermeldung ohne vorherigen Requester. Nochmaliges Eingeben von »remreq« versetzt das System wieder in den alten Zustand. Wenn Sie Teile der

In der Kürze liegt die Würze

»startup-sequence« gegen Requester sichern wollen, umschließen Sie diese einfach mit einem Aufruf von »remreq« vor und nach den kritischen Befehlen. Das Listing ist so kurz, daß wir Ihnen eine ausführliche Erläuterung nicht vorenthalten können: Normalerweise existiert zu jedem im CLI gestarteten Prozeß ein Fenster, für die Standard Ein- und Ausgabe tätigt. Dazu befindet sich in der Prozeß-Struktur eine Komponente, die auf das entsprechende Fenster für einen Requester zeigt oder für das übliche CLI-Fenster mit Null initialisiert ist. Um diese zu finden, wird die Exec-Funktion FindTask aufgerufen, die einen Zeiger auf unsere Task-Struktur ermittelt. Da die Task-Struktur vollständig in der Prozeß-Struktur enthalten ist, ist der zurückgegebene Wert gleichzeitig auch der Zeiger auf unseren Prozeß. Das machen wir dem Compiler explizit mit einer Typenumwandlung klar:

```
(ThisProc=(struct
Process *)FindTask(0L))
```

Mit einem kleinen Trick gaukeln wir dem Betriebssystem nun vor, daß die Benutzung eines Requesters nicht mehr möglich ist: Die Komponente »pr_WindowPtr« bekommt den Wert -1 zugewiesen, so daß unser Prozeß (das CLI) annimmt, daß kein Fenster mehr für entsprechende Zwecke vorhanden ist. Um aber den alten Wert später wiederverwenden zu können, kopieren wir ihn vorher in ein passendes Element der Task-Struktur, das für den Benutzer zur freien Verfügung steht (»ThisProc->pr_Task.tc_UserData«). Bei einem erneuten Aufruf von »remreq« passiert dann genau das Gegenteil; der »pc_WindowPtr« wird restauriert.

Da natürlich jeder Prozeß eine eigene Prozeß-Struktur hat, ist konsequenterweise die Wirkung von »remreq« nicht global, sondern auf den aktiven Prozeß beschränkt. Das ist nicht unbedingt ein Nachteil, da dieser Befehl seinen Sinn gerade durch den gezielten Einsatz erhält.

(Christian Wolf)

Programmname:	RemReq
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztek oder Lattice
Aufrufe:	siehe Text

```
Programmautor: Christian Wolf
-----------------------------
us0 #include <libraries/dosextens.h>
2i  #ifdef LATTICE
Wa  #include <proto/all.h>
m3  #else
fJ  #include <functions.h>
NJ  #endif
oA  main()
4X  {
en1   struct Process *ThisProc=(struct Process *)FindTask(OL);
Zd    if (ThisProc->pr_WindowPtr != -1)
7a5      {
zr6        ThisProc->pr_Task.tc_UserData=ThisProc->pr_WindowPtr
           ;
q1         ThisProc->pr_WindowPtr = -1;
Ej5      }
4r1   else
Cf5      {
Ih6        ThisProc->pr_WindowPtr=ThisProc->pr_Task.tc_UserData
           ;
rC         ThisProc->pr_Task.tc_UserData=0;
Jo5      }
Kp0 }

(C) 1989 M&T
```

Listing 1. »remreq.c« verbannt die System-Requester

Aus unserem Bücher-Regal

Wer die Fähigkeiten des Amiga ausnutzen möchte, braucht eine gute Anleitung. Die Handbücher zum Amiga und zu den verschiedenen Programmiersprachen beschränken sich meist auf den Hinweis, sie seien nicht als Lehrbuch gedacht. Viele Bücher erheben den Anspruch, diese Lücke zu schließen. Lesen Sie auf den folgenden Seiten, welche für Sie richtig sind.

Programmierpraxis Intuition

»Es gibt zu viele Bücher über den Amiga, die viel zuwenig behandeln.« Um mit dieser Unsitte aufzuräumen, ist Peter Wollschlaeger mit seinem Buch »Programmierpraxis Intuition« angetreten – und er erledigt dies gründlich.

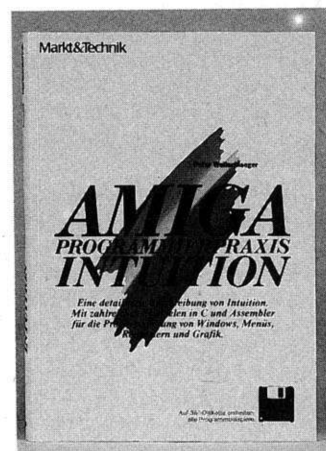
Der Amiga steht in dem Ruf, ein ausgesprochen kompliziertes Betriebssystem zu haben. Die zugehörige Dokumentation umfaßt vier dicke Bände. Während andere Computer oft nicht mehr als einen Kommando-Interpreter bieten, kann der Amiga weit mehr. Arbeiten in mehreren Fenstern gleichzeitig, Programmsteuerung mit Maus und Menüs sind selbstverständlich.

Aber leichte Bedienbarkeit und Multitasking haben auch Nachteile. Als Programmierer müssen Sie sich beim Amiga um weit mehr kümmern als bei einfacheren Computern. Sie müssen eine große Anzahl Strukturen und Funktionen kennen und wissen, wie diese angewendet werden.

Diese für alle Teile des Betriebssystems zu beschreiben, würde entweder zahlreiche Bände füllen oder unvollständig bleiben. Peter Wollschlaeger wählt in seinem Buch eine elegante Alternative. Er beschränkt sich auf einen überschaubaren Teil des Gesamtsystems und bietet für diesen eine übersichtliche und solide Beschreibung.

In »Programmierpraxis Intuition« finden Sie eine ausführliche Anleitung zur Programmierung der Amiga-Benutzeroberfläche. Voraussetzung sind Kenntnisse von C oder Assembler. Wer in einer dieser Sprachen programmiert, wird sich in diesem Buch schnell zurechtfinden. Es geht auf alle Aspekte der Intuition-Programmierung ein und bietet zu jedem Gebiet Beispielprogramme, die auf Diskette dem Buch beiliegen.

Angenehm fällt auf, daß in C nur Routinen verwendet wurden, die sowohl mit dem Aztec- als auch mit dem Lattice-Compiler zusammenarbeiten. Assembler-Programmierer finden am Ende jedes Kapitels eine

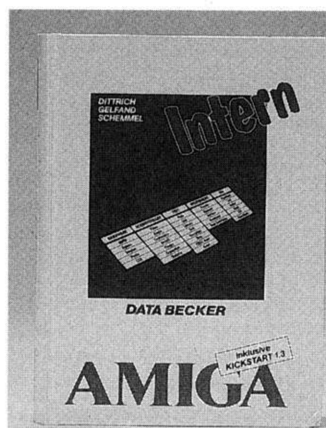


Umsetzung der wichtigsten Beispielprogramme.

Nach einer Einführung, die intensiv auf die Grundlagen der Intuition-Programmierung eingeht, erfährt der Leser alles Wissenswerte über die Programmierung von Screens, Windows und der dazugehörigen Bedienungselemente (Menüs, Gadgets und Requester). Der Anhang macht das Buch zu einem Nachschlagewerk, das man auch nach langen Programmierstunden nicht aus der Hand legt. Hier sind alle verwendeten Datenstrukturen nochmals in C- und Assemblerschreibweise dokumentiert.

»Programmierpraxis Intuition« ist für jeden, der sich mit diesem Thema auseinandersetzt, ein unbedingtes Muß. *(so)*

Peter Wollschlaeger, Programmierpraxis Intuition, M&T Verlag, 328 Seiten, ISBN 3-89090-593-5, Preis 69 Mark



Amiga Intern

Der Name »Intern« ruft bei Umsteigern vom C64 Erinnerungen wach. Mit dem »64 Intern« setzte Data Becker Maßstäbe. Wird Amiga Intern im zweiten Anlauf diesem Anspruch gerecht?

Die erste Auflage von Amiga Intern bezog sich nur auf einen Teilbereich des Amiga-Betriebssystems. Die zweite Auflage erscheint schon äußerlich stark verändert: Der erste Band (der mit der ursprünglichen ersten Auflage teilweise identisch ist) enthält jetzt nur noch die Bereiche Hardware, Exec und AmigaDOS. Zusätzlich wurde er durch einen zweiten Band zum Thema Libraries und Devices ergänzt.

Dieses Konzept kommt dem komplizierten Betriebssystem des Amiga weit besser entgegen, als dies in der ersten Auflage der Fall war. Im ursprünglichen »Intern« wurden wichtige Teilbereiche nur kurz angerissen. Dagegen erhält der Programmierer mit der zweiten Auflage grundlegende Informationen zu allen Bereichen des Betriebssystems.

Außer einigen Ergänzungen (besonders zum Amiga 2000) wurde das erste Kapitel zum Thema Hardware fast unverändert übernommen. Angesichts der Qualität der ersten Auflage in diesem Bereich ist diese Entscheidung sicher gerechtfertigt.

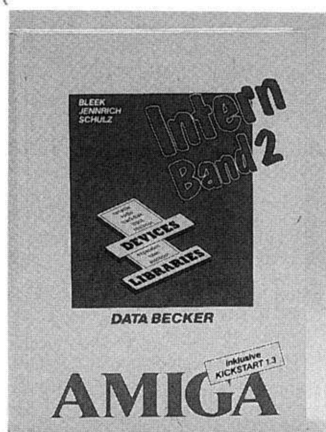
Alle anderen Kapitel wurden dagegen sehr gründlich überarbeitet:

So ist ein eigenes Kapitel mit Hinweisen zur »sauberen« Programmierung unter Multitasking-Bedingungen enthalten. Der »Exec«-Teil wurde um je ein Kapitel zu sogenannten »Semaphores« und zum Booten aus der RAM-Disk erweitert.

Der AmigaDOS-Teil wurde komplett umgestaltet. Er umfaßt jetzt endlich eine vollständige Dokumentation aller für den Programmierer wichtigen Daten wie DOS-Bibliothek, Standard-Ein-/Ausgabe, Handler, Devices und Disk-Format. Der interne Aufbau eines AmigaDOS-Programms – einschließlich Code-Segmentierung und Hunks – wird ausführlich beschrieben. Außerdem wird auf die Neuerungen der Kickstart 1.3 eingegangen (beispielsweise Fast-Filing-System).

Neu in der zweiten Auflage sind die Kapitel über »Expansionsarchitektur« und »Januskonzept«. Während ersteres wohl eher für Hersteller von Erweiterungshardware interessant ist (für diese aber viel zu knapp), beschreibt das zweite die Interna der Schnittstelle zur MS-DOS-Welt, des »Bridgeboards«. Wer eine Brückenkarte im Amiga 2000 besitzt, findet hier – neben dem »Amiga System-Handbuch« – die einzige deutschsprachige Beschreibung dieser Schnittstelle.

»Intern Band 2« ist mehr als nur die Fortsetzung von Band 1. Nachdem die – vorher viel zu knapp gehaltenen – Kapitel über Libraries und Devices aus Band 1 herausgefallen sind, ist diesem wichtigen Thema ein eigener Band von über 800 Seiten gewidmet. Natürlich ist das Intern kein Lehrbuch für Systemprogrammierer. Dennoch sind im ersten Kapitel viele nützliche Hinweise zu Programmier-Konventionen unter Multitasking enthalten. Die anderen Kapitel sind eher als Nachschlagewerk zu den einzelnen Betriebssystemteilen aufgebaut.



Beispielsweise beschreiben die ersten 250 Seiten ausführlich die Programmierung der Devices anhand von Beispielen mit Parallel-, Serial-, Printer-, Gameport-, Input-, Timer- und Trackdisk-Device.

Nach einer kurzen Beschreibung des IFF-Formats und seiner Varianten für Grafik, Text, Musik und Samples geht es in die Vollen: Fast 500 Seiten sind den Libraries gewidmet. Natürlich können die Amiga-Libraries auch bei diesem Umfang nicht abschließend behandelt werden. Dennoch wird für jede Routine einschließlich der neuen Libraries der Kickstart 1.3 (beispielsweise »Romboot«-Library) in einer Kurzübersicht die Syntax und die Funktion der jeweiligen Routine beschrieben. Besonders hilfreich sind die Überschriftsbalken mit dem Namen der Routine und einer Stichwort-Beschreibung der Aufgaben (beispielsweise »Allocate: Belegt einen Speicherblock«).

Das letzte Kapitel geht auf die Preferences-Datenstruktur ein. Neben einer Aufstellung der von Preferences verwalteten Daten (und der Beschreibung des Zugriffs auf diese Daten) sind besonders die Tips zu Druckertreibern und Fonts interessant.

Beide Bände des Intern sind jeder für sich ein unabhängiges Nachschlagewerk, in dem Sie immer wieder neue interessante Anregungen für eigene Experimente finden werden. Auf den ersten Blick erscheint der Preis von insgesamt fast 140 Mark etwas hoch, aber diese geballte Information auf 1500 Seiten finden Sie in keinem anderen Werk. Wer sich für die Programmierung mit Hilfe der Systemroutinen interessiert, sollte dieses Werk nie weiter als eine Armlänge von der Tastatur entfernt aufstellen. *(so)*

Dittrich, Gelfand, Amiga Intern Band 1, Schemmel, Data Becker, 716 Seiten, ISBN 3-89011-104-1, Preis 69 Mark

Bleek, Jennrich, Schulz, Amiga Intern Band 2, Data Becker, 880 Seiten, ISBN 3-89011-268-4, Preis 69 Mark

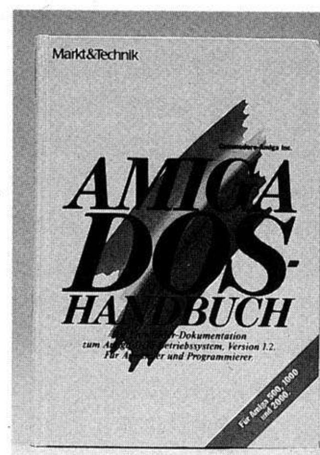
AmigaDOS-Handbuch

AmigaDOS ist mehr als nur das für jeden sichtbare CLI. Wer sich entsprechend informiert, kann mit Hilfe dieses Betriebssystemteils komfortabel auf Diskette zugreifen, eigene »Prozesse« in der Multitasking-Umgebung installieren und vieles mehr.

Die Fakten soll dieses Buch vermitteln. Es handelt sich um die Übersetzung der englischen Entwicklerdokumentation zu AmigaDOS. Diese besteht im Original aus drei Bänden. In der deutschen Version wurden alle drei zu einem Band zusammengefaßt.

Der erste Abschnitt enthält alle wichtigen Informationen zum CLI. Der Einsteiger findet alles Wissenswerte zur Bedienung der Ein- und Ausgabegeräte (Laufwerke, Tastatur, Bildschirm, etc.). Die Übersicht zu den CLI-Befehlen stellt jeden einzelnen Befehl kurz vor, erklärt die Wirkungsweise und die Anwendung anhand von Beispielen. Am Ende befindet sich eine Kurzreferenz zu allen CLI-Kommandos und eine Beschreibung der mitgelieferten Editoren »Ed« und »Edit«.

Neben der Benutzeroberfläche bietet AmigaDOS eine Vielzahl von Routinen, die vom Programmierer genutzt werden können. Diese befinden sich in der »dos.library« und können von fast allen Programmiersprachen angesprochen werden. Im zweiten Buchteil werden zunächst Hinweise zur Pro-



grammierung unter AmigaDOS gegeben. Darauf folgt eine Darstellung aller AmigaDOS-Routinen und deren Anwendung in eigenen Programmen. Eine Beschreibung des Metacomco-Assemblers und des »ALink«-Linkers bilden den Abschluß.

Das dritte Buch heißt im Original »Technical Reference Manual« und geht sehr tief auf die Details der Implementierung dieses Betriebssystemteils ein. Dieser Teil klingt zweifellos selbst für fortgeschrittene Programmierer nach böhmischen Dörfern. Hier werden neben dem Aufzeichnungsformat der Diskette auch die Struktur von AmigaDOS-Prozessen und »Hinweise für den fortgeschrittenen Entwickler« behandelt.

Die Dreiteilung dieses Buches ist sicher auf den ersten Blick problematisch. Denn durch die überaus große thematische Spannweite ist für jede Zielgruppe stets nur ein Ausschnitt um etwa einem Drittel der Seiten interessant. Der Rest wird schnell als langweilig oder für zu schwierig eingestuft. Andererseits ist dies ein Buch, das Sie auf Ihrem Weg vom CLI-Einsteiger bis zum

Profi-Programmierer begleitet. Sicher werden Sie immer nur einen Teil als Nachschlagewerk benutzen, aber dieser ist enorm hilfreich. *(so)*

Commodore-Amiga Inc., AmigaDOS-Handbuch, M&T Verlag, 342 Seiten, ISBN 3-89090-465-3, Preis 59 Mark

Das große C-Buch

Nachdem ich zum erstenmal ein C-Programm gesehen hatte, war mein Eindruck ein vernichtender: Wer C lernen will, sollte viel Geduld und Verständnis für pharaonische Schreibweisen mitbringen. Vergessen Sie's.

Wenn Sie Hieroglyphen verwenden wollen, ist das Ihre Sache. Es geht jedenfalls auch ohne. C ist eine Programmiersprache wie jede andere, nur kürzer in der Ausdrucksweise. Sie kommt also der Spezies von Programmierern entgegen, die ellenlange Ausdrücke verabscheuen.

Diese werden auf dem Amiga zusätzlich unterstützt: Eine Vielzahl von Systemroutinen können vom Programmierer genutzt werden (warum das Rad noch einmal erfinden?). C überzeugt bei der einfachen Nutzung dieser Routinen.

»Das große C-Buch« ist kein Buch für den Einsteiger – auch nicht für den sehr engagierten. Sie sollten also die Sprachelemente von C bereits beherrschen, bevor Sie dieses Buch aufschlagen; denn dieses verliert keine Zeit mit langen Vorreden und Erklärungen – es geht sofort ans Eingemachte.

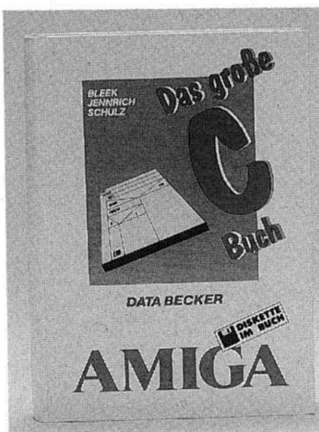
Die ersten beiden Kapitel geben einen tiefen Einblick in die Funktionsweise des Aztec-Compilers. Die Autoren wollen hiermit aber keineswegs einen Ersatz für das Handbuch bieten. Vielmehr liefert dieser Teil zahlreiche Tips und Tricks für die professionelle Nutzung dieses Compilers.

Das folgende Kapitel ist eine wahre Fundgrube für jeden Intuition-Programmierer. Es behandelt alle Aspekte dieses Betriebssystemteils, angefangen von Screens und Windows über Menüs und Gadgets bis zu Speicherverwaltung und Messagesystem. Jeder Bereich wird durch detaillierte Anwendungsbeispiele »gewürzt«, die den Stoff leichter verdaulich machen.

Den krönenden Abschluß bildet ein ausführliches Projekt, das von der Planungsphase bis zum fertigen Programm beschrieben wird. Ein selbstgebauter Editor ist der Lohn der Mühe, wobei das hier gewonnene Wissen weit über den vorher behandelten Bereich hinausgeht. Übrigens scheinen die Autoren vom Konzept dieses Editors sehr überzeugt gewesen zu sein: Der »EdWork«, der von Data Becker als kommerzielles Programm vertrieben wird (siehe Editorentest auf Seite 8), basiert größtenteils auf diesem Projekt.

»Das große C-Buch« ist für den fortgeschrittenen Programmierer ein erstklassiges Lehrbuch zur Betriebssystem-Programmierung. Die wichtigsten Techniken werden anhand interessanter Beispiele – zumeist aus dem Bereich Intuition – anschaulich erklärt. *(so)*

Bleek, Jennrich, Schulz, Das große C-Buch, Data Becker, 681 Seiten, ISBN 3-89011-191-2; Preis 69 Mark



Amiga Assembler-Buch

Es gibt genügend Programmierer, die den Sinn von Assembler für den Amiga grundsätzlich in Frage stellen. Fast alle Probleme lassen sich ebenso gut (und anschaulicher) in einer Hochsprache lösen. Dennoch sind die »Bitfledderer« noch nicht ausgestorben.

Dies liegt zum einen an den Umsteigern von anderen Computern, die Assembler als einzige Möglichkeit kennen, ihren Computer wirklich auszureizen. Zum andern ist für viele Einsteiger ein C-Compiler zu teuer.

Ein Buch, das sich an den engagierten Einsteiger ebenso richtet wie an den Umsteiger von anderen Computern, ist das Amiga Assembler-Buch von Peter Wollschlaeger. Nach drei theoretischen Kapiteln zur Assembler-Programmierung geht es »ganz schnell zur Praxis«. Anhand kurzer Beispiele zeigt der Autor die Lösung typischer Programmierprobleme in Assembler (Verzweigungen, Schleifen, Ein- und Ausgabe).

Nachdem das Handwerkszeug erarbeitet ist, geht es sofort in die Vollen: Der komplette 68000er-Befehlssatz wird beleuchtet, die wichtigsten Bereiche des Betriebssystems werden kurz gestreift und ein Kapitel befaßt sich ausführlich mit dem Einbinden von Assembler-Routinen in Basic.

Der Anhang besteht neben einer kompletten Befehlsliste aus einer Zusammenstellung der sogenannten »Library Vector Offsets«, die zum Aufruf der Betriebssystemroutinen benötigt werden, sowie einer Kurzreferenz der Funktionen und Datenstrukturen.

Das Assembler-Buch ist derzeit das beste Lehrwerk für alle, die den Einstieg in die Assembler-Programmierung wagen möchten. Wollschlaeger macht sehr deutlich, in welchen Situationen Assembler sinnvoll ist und wo nicht. Der Einsteiger wird das Buch auch dann nicht aus der Hand legen, wenn er sich längst als Fortgeschrittener oder Profi betrachtet. Allerdings ist zum Verständnis nicht nur aufmerksames Lesen, sondern eine Menge Übung notwendig. Auch kommt man bei den Kapiteln zur Intuition- und Exec-Programmierung nicht ohne zusätzliche Informationen aus. Eine umfassende Behandlung dieser Themen hätte den Rahmen des Buches gesprengt.

Wer seinen Amiga in Assembler programmieren möchte und Erfahrungen in einer anderen Sprache mitbringt, kann zur Zeit kein besseres Buch finden. Besonders die einfache, lockere Ausdrucksweise ist bei Büchern zu diesem Thema eine echte Rarität.

Der Assembler-Kurs von Seite 74 basiert auf diesem Buch. Die Beispiel-Listings sind dem Buchmaterial entnommen. *(so)*

Peter Wollschlaeger, Amiga Assembler-Buch, M&T Verlag, 329 Seiten, ISBN 3-89090-525-0, Preis 59 Mark



Amiga-Programmierhandbuch

Jeder erfahrene Amiga-Programmierer weiß, daß ein Buch von knapp 400 Seiten nicht alle Geheimnisse des Amiga enthüllen kann. So gesehen sollte man mit dem Titel »Programmierhandbuch« für solch ein dünnes Bändchen ruhig etwas vorsichtig umgehen – für einen kleinen Hinweis auf den tatsächlichen Inhalt wäre der potentielle Käufer sicher dankbar.

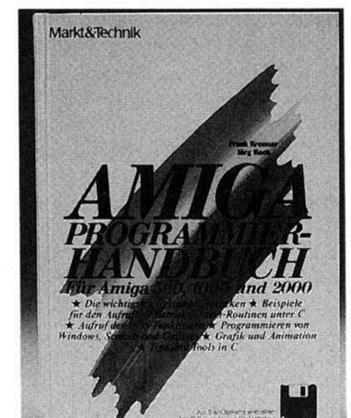
Hat man sich damit abgefunden, daß auch dieses Buch – erwartungsgemäß – nur einen Ausschnitt des Betriebssystems behandelt, findet man im Innern so manches Goldstück:

Das Programmierhandbuch ist ein umfassendes Nachschlagewerk zu Intuition. Die »Intuition.library«, die »Graphics.library« und die mathematischen Funktionen werden auf ähnliche Weise behandelt wie im »Amiga Intern«, allerdings ausführlicher.

Einem Kapitel zu Prinzipien der Programmierung in C folgen die Themen Screens und Windows, Grafikroutinen, Programmbedienung und Ein-/Ausgabe. Ein abschließender Sonderteil enthält »Vermischtes« zu Themen anderer Bereiche. Diese sind allerdings zu kurz beschrieben, um wirklich hilfreich zu sein.

Einen guten Eindruck hinterläßt das Konzept, ein Nachschlagewerk zu bieten, das durch ausführliche Programmbeispiele aufgelockert wird. Das Amiga-Programmierhandbuch ist ein Nachschlagewerk zu den Themen Intuition und Grafik. Es ist schwer verdaulich für den Intuition-Anfänger, aber als begleitende Literatur beispielsweise zu »Programmierpraxis Intuition« gut geeignet. *(so)*

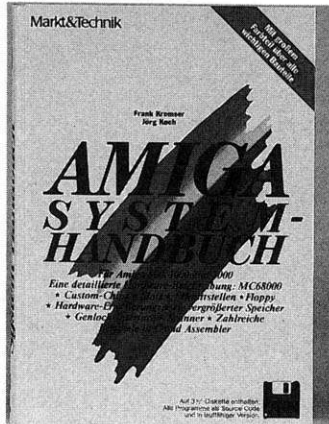
Frank Kremser und Jörg Koch, Amiga-Programmierhandbuch, M&T Verlag, 387 Seiten, ISBN 3-89090-491-2, Preis 69 Mark



Amiga System-Handbuch

Obwohl es den Amiga schon einige Zeit gibt, sind Bücher, die seine Hardware beschreiben, eine Seltenheit. Das System-Handbuch ist so ein Hardware-Buch. Die Technik des Amiga wird umfassend und genau dokumentiert. Was ebenso wichtig ist: Die Programmierung der Hardware wird mit zahlreichen Programmbeispielen in C und Assembler demonstriert.

Die Autoren beginnen mit einem (fast historischen) Abriß der Amiga-Entwicklung vom »alten« A 1000 über den A 500 bis zum Modell A 2000B. Nach einer grundsätzlichen Erläuterung der Architektur dieser Systeme folgen genaue Beschreibungen des Bootvorgangs, des Prozessors und der Custom-Chips. Mit einem Intensivkurs über Blitter-, Copper- und Audio-Hardware wird der Leser zum Kenner der Grafik- und Musiktechnik. Aber auch die unterschiedlichen Steckplätze der Amiga-Modelle sowie die gesamte Kommunikation mit der Außenwelt werden durchleuchtet. Dazu gehören Schnittstellen zu den Diskettenlaufwerken, die serielle und parallele Schnittstelle, Tastatur und Maus, die MS-DOS-Hardware wie Sidecar oder PC-Karte, mögliche RAM-Erweiterungen



und die Monitoranschlüsse. Dabei beschreibt das Autorenduo die beteiligten Hardwarekomponenten detailliert. Die Programmbeispiele behandeln schwerpunktmäßig die unmittelbare Programmierung der Systemregister, aber auch auf der Betriebssystemebene wird die Anwendung der Hardware eindrucksvoll und doch überschaubar demonstriert. Ergänzt werden die Beschreibungen durch viele Grafiken und Farbfotos der Amiga-Chips und Systemplatinen. Gekrönt haben die Autoren ihr Werk mit Bastelanregungen vom »einfachen« PAL-Modulator bis hin zu einem Genlock-Interface oder einem Digitizer mit Schaltplänen und Bestückungslisten sowie einer vollständigen Beschreibung der Janus-Library und der Expansion-Library.

Dieses Buch ist eine Fundgrube für jeden, der sich mit der Hardware des Amiga beschäftigt. Wer die Beispielprogramme auf der beiliegenden Diskette startet, bekommt einen Eindruck davon, was man auf und mit dem Amiga alles machen kann, wenn man die Möglichkeiten des Systems nur richtig zu nutzen weiß.

Georg Zweschper/so

Koch/Kremser, Amiga System-Handbuch, Markt&Technik, 421 Seiten, Preis 79 Mark

Amiga – Eine Einführung in die Assemblerprogrammierung

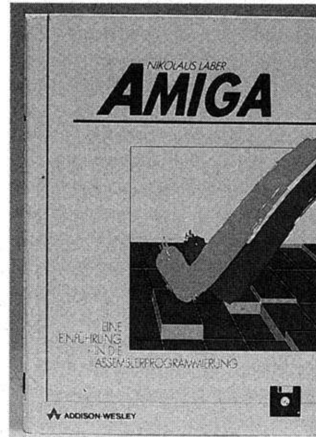
Ein Buch, das mit dem Wort »Einführung« im Titel wirbt, spricht in erster Linie Einsteiger an. Diese sind jedoch mit dieser »Einführung« restlos überfordert.

Nikolaus Laber setzt beim Leser eine Menge voraus: Dieser sollte mit den Grundlagen der Digitaltechnik vertraut sein, mindestens eine Programmiersprache beherrschen und den Amiga gut kennen. Wenn er außerdem etwas Assembler-Erfahrung mitbringt, ist dieses Buch bestens für ihn geeignet.

Bereits vom ersten Kapitel an (»Die Hardware«) wird ein atemberaubendes Tempo vorgelegt. Das kommt Lesern mit einigen Erfahrungen ungemein entgegen; sie brauchen nicht – wie in vielen anderen Büchern – die Seiten mit längst bekannten Grundlageninformationen überblättern. Die wichtigsten Grundlagen wie Hardware, Aufbau des Betriebssystems oder Bedienung des Assemblers werden in kompakter Form auf den ersten 130 Seiten abgehandelt.

Im fünften Kapitel wird der Leser mit der Assemblerprogrammierung des Amiga vertraut gemacht. Die Beispielprogramme kommen fast ausschließlich aus dem Bereich der Programmierung mit Intuition. Der Zugriff auf andere Betriebssystembereiche sollte jedoch mit dem hier erarbeiteten Wissen und einigen Informationen aus ergänzender Literatur kein Problem mehr sein.

Den Abschluß bilden die – besonders für Seka-Programmierer wichtigen – Offsetlisten der Systembibliotheken. Hier finden Sie alle Routinen einschließlich der Registerbelegung.



Die »Einführung in die Assemblerprogrammierung« ist ein sehr gutes Lehrbuch. Es richtet sich jedoch keinesfalls an den Einsteiger, Sie sollten beim Erwerb dieses Buches bereits einiges an Grundwissen mitbringen. Wenn Sie beispielsweise den Assemblerkurs in diesem Heft intensiv bearbeitet haben, ist dieses Buch genau richtig für Ihre »Fortbildung«.

(so)

Nikolaus Laber, Amiga – Eine Einführung in die Assemblerprogrammierung, Addison-Wesley, ISBN 3-89019-145-3, 324 Seiten, Preis 68 Mark

Systemprogrammierung in C

Neben der inzwischen in immer größerer Zahl erscheinenden deutschen Amiga-Literatur sind die Übersetzungen englischsprachiger Werke für Amiga-Anwender und -programmierer interessant. Zu dieser Sparte zählt auch »Systemprogrammierung in C« (Originaltitel: Inside the Amiga) von John Thomas Berry.

In der »Einführung« findet der Programmierer eine Kurzbeschreibung von Hardware und Betriebssystem des Amiga sowie einige Anmerkungen zum Aufbau des Buches. »Die Programmierung des Amiga« stellt Arbeitsweise und Anwendung eines C-Compilers (Lattice) innerhalb der Systemumgebung des Amiga und einige spezielle Datentypen vor, die bei der Programmierung zum Einsatz kommen. »Wie man Intuition benutzt« – dieser Titel spricht für sich: Hier werden die wichtigsten Intuition-Elemente (Screens, Fenster, Menüs, Gadgets, Requester) und deren Programmierung vorgestellt. Dabei geht der Autor auch recht ausführlich auf die Programmierung von Ein-/Ausgabeoperationen (IDCMP, console.device) unter Intuition ein. Das Kapitel »Prozeßkontrolle und Amiga-DOS« bietet einen Einblick in das Multitasking-Betriebssystem des Amiga und erklärt die Grundbegriffe seiner Programmierung. Für den erfahrenen Programmierer ist diese Erklärung ausreichend; der Anfänger wird sich damit schwer tun.

»Zeichnen mit Intuition« erklärt die Zeichenfunktionen, die dem Programmierer unter Intuition zur Verfügung stehen. Zusätzlich werden noch einige der Grafikfunktionen der Grafikbibliothek des Amiga vorgestellt. »Sprites bewegen« behandelt alle nötigen Funktionen und Strukturen, um Sprites zu definieren und in eigenen Programmen einzusetzen.

Mit »Die Programmierung von Klängen« demonstriert der Autor die Verwendung des Amiga zur Klangerzeugung. Dabei werden erst einige allgemeine Grundlagen zur Erzeugung von Klängen und danach die Amiga-Programmierung im speziellen vorgestellt. Der Abschnitt »Künstliche Sprache« widmet sich der Sprachausgabe mit den im Amiga zur Verfügung stehenden Möglichkeiten. »Programmierung mit Disketten-Dateien« bildet den Abschluß des Buches.

Natürlich kann auf etwas über 400 Seiten nur ein kleiner Ausschnitt des Amiga-Betriebssystems behandelt werden. Das Buch ist trotz der in einigen Kapiteln zu kurz geratenen Funktionsbeschreibungen engagierten C-Programmierern zu empfehlen. Sie werden die in diesem Werk zusammengefaßten und damit schnell verfügbaren Informationen bei der Programmierung schätzen.

(Ingolf Krüger/so)

John Thomas Berry, Systemprogrammierung in C, Tewi Verlag GmbH, 454 Seiten, ISBN 3-921803-71-3, Preis 59 Mark



Das Doppellaufwerk ist vergeben

Die Umfrage aus dem Sonderheft 2 hat viele neue Anregungen gebracht. Zahlreiche Leser haben uns ihre Meinung mitgeteilt, wofür wir herzlich danken. Unter allen Einsendungen haben wir den Gewinner des Doppellaufwerks von Soyka Datentechnik ausgelost. Lesen Sie, welche Ergebnisse die Umfrage erbracht hat und was unsere Leser von uns erwarten.

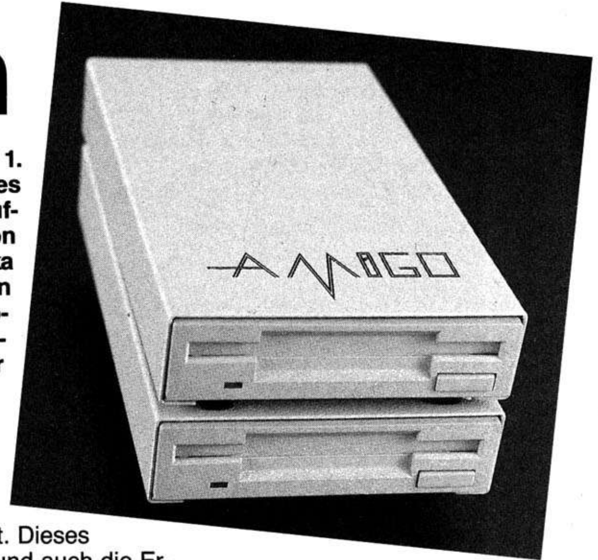


Bild 1. Dieses Doppellaufwerk von Soyka steht nun bei Thomas Steininger

Der Gewinner



Thomas Steininger heißt der strahlende Gewinner bei unserer Leserumfrage. Thomas ist 15 Jahre alt und wohnt in Allershausen. Er besucht zur Zeit die 8. Klasse der Realschule Freising. Am Informatikunterricht nimmt er besonders intensiv teil. Hier lernt er alle Grundlagen für die Programmierung des Amiga, den er seit fast zwei Jahren besitzt.

Mit Amiga-Basic begann er, heute ist ihm diese Sprache zu langsam. Mit dem Umstieg auf Modula will Thomas die Leistungsfähigkeit des Amiga voll nutzen. Neben der Programmierung beschäftigt er sich mit Grafik- und Musikprogrammen und kleinen Hardware-Basteleien.

Thomas freut sich sehr über das Doppellaufwerk, mit dem er jetzt noch professioneller mit dem Amiga arbeiten kann.

Im Sonderheft 2 hatten wir eine Bitte an Sie gerichtet: Wir fragten nach Ihrer Meinung zum AMIGA-Sonderheft. Sie hatten dadurch die Chance, sich aktiv an der Gestaltung der Sonderhefte zu beteiligen. Zahlreiche Leser haben diese Möglichkeit genutzt. Die statistische Auswertung der Umfrage ist inzwischen abgeschlossen, hier einige der Ergebnisse:

Bei der Selbsteinschätzung der Einsender erlebten wir eine kleine Überraschung: 11 Prozent aller Einsender bezeichnen sich als Einsteiger ohne Vorkenntnisse. Wir hatten diesen Anteil etwas geringer eingeschätzt. Als Einsteiger bezeichneten sich 35 Prozent aller Leser. Die größte Gruppe bilden mit knappen Vorsprung die Fortgeschrittenen mit 39 Prozent. Auf erfahrene Fortgeschrittene und Profis entfallen 15 Prozent. Dieses Ergebnis und auch die Erfahrungen aus zahlreichen Telefongesprächen und Briefen zeigt: Es ist sehr wichtig, Grundlagen noch genauer zu erklären, damit auch die Einsteiger viel Freude an den Sonderheften haben. Diese Aufgabe nehmen wir sehr ernst.

Der richtige Weg

Die Bewertung der Sonderhefte 1 und 2 zeigt, daß wir auf dem richtigen Weg sind: 65 Prozent gaben beim Kriterium »informativ« die Noten 1 und 2, 64 Prozent vergaben diese Note für das Merkmal »verständlich«. Trotz dieser guten Bewertung werden wir weiter bemüht sein, auch das restliche Drittel voll zufriedenzustellen.

Bei dem Preis von 16 Mark pro Sonderheft hagelte es dann schlechte Noten. 20 Prozent gaben hier die Note 5, 24 Prozent sogar eine 6. Natürlich verstehen wir, daß 16 Mark im Gegensatz zum Preis für das monatlich erscheinende AMIGA-Magazin (7 Mark) sehr viel erscheint. Aber bedenken Sie: Im Sonderheft bekommen Sie auf durchschnittlich 150 Seiten (redaktionelle Beiträge, Gesamtseitenzahl 164) geballte Informationen zu einem oder zwei Themenschwerpunkten. Zudem finden Sie im Sonderheft häufig wesentlich umfangreichere Programme, als sie beispielsweise das AMIGA-Magazin veröffentlichen kann.

Bei den Themen, die wir Ihrer Meinung nach im Sonderheft berücksichtigen sollen, stehen Tips & Tricks, Grafik und Programmieren ganz vorn.

Die Auswertung aller Einsendungen hat zu vielen neuen Ideen geführt, die wir in den kommenden Heften umsetzen werden. Bereits in diesem Sonderheft finden Sie eine völlig neue Rubrik: den Workshop.

Unter dieser Rubrik werden wir intensive Hilfestellung zur bestmöglichen Ausnutzung einzelner Programme geben. Schritt für Schritt lernen Sie so den effektiven Einsatz eines Anwendungsprogramms, einer Programmierhilfe (z.B. in diesem Heft PowerWindows) oder eine Verknüpfung mehrerer PD-Programme kennen.

Natürlich verändern sich die Interessen, wenn der Amiga vertrauter wird. Wir möchten Sie auch weiterhin bitten, uns Ihre Meinung mitzuteilen.

Die Sonderhefte-Redaktion freut sich auf Ihre Anregungen.

(rs)

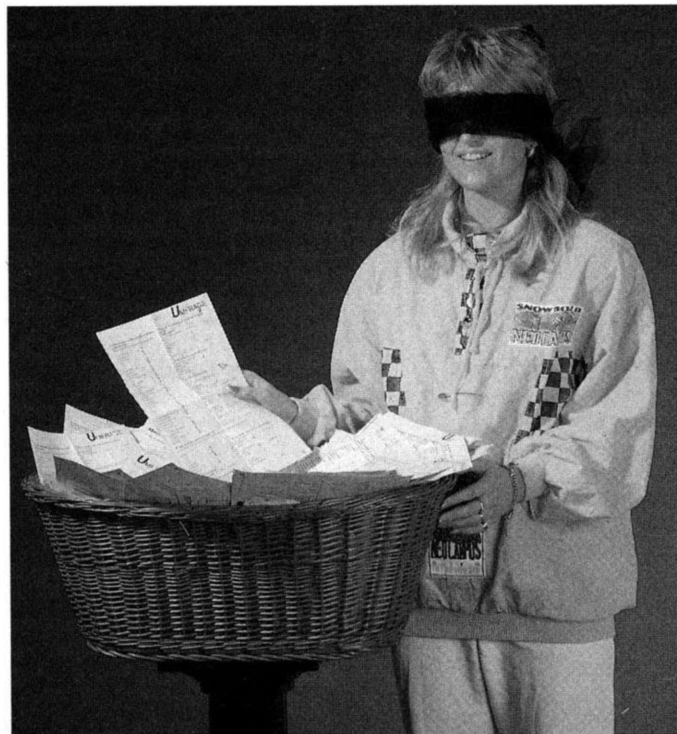


Bild 2. Unsere Redaktionsassistentin Sylvia Derenthal als Glücksfee. Sie zog den Fragebogen von Thomas Steininger aus allen Einsendungen heraus.

Programmabbruch in Assembler

Bei manchen Assembler-Programmen ist nach Ausführung eine sofortige Rückkehr in das System oder, wenn das Programm vom Assembler aus gestartet worden ist, in den Assembler nicht wünschenswert. Vielmehr möchte man sich vielleicht am Bildschirm die Resultate der Programmausführung, eine Grafik oder ähnliches, erst in Ruhe anschauen. Irgendwann sollte das Programm allerdings zurückkehren. Die folgende Routine zeigt ein Beispiel mit einer möglichen Lösung dieses Problems. Aus der Adresse $dff00a wird die Position des Mauszeigers auf dem Bildschirm gelesen und in das Bildschirmfarbregister geschrieben.

```
LOOP: MOVE $DF00A,$DFF180
      ANDI.B #64,$BFE001
      BNE LOOP
      RTS
```

Durch Bewegen der Maus kommt nun auf dem Bildschirm Farbe ins Spiel. Der eigentliche Kern — das Testen der Speicherstelle, in der ein Druck auf die linke Maustaste registriert wird — mit dem ANDI-Befehl beendet das Programm nach einer entsprechenden Betätigung der Taste. *(Martin Dencker/rs)*

Mit beiden Armen tief im System

Hier ein paar kleine Routinen für die Assembler-Freaks: Abfrage der rechten Maustaste:

```
LOOP: BTST #10,$DFF016
      BNE.S LOOP
```

Power-LED dunkel:

```
      OR.B #2,$BFE001
```

Power-LED hell:

```
      ANDI.B #253,$bfe001
```

Unterroutine für die Initialisierung einer eigenen Copperliste:

```
set_user_copperlist:
         MOVE.L 4,A6
         LEA Gfxname,A1
         JSR -408(A6)
         MOVE.L D0,A5
         ADD.L #$32,A5
         MOVE.L A5,savecop2
         MOVE.L (A5),savecop
         MOVE.L #COPLIST,(A5)
         RTS
```

Unterroutine für das Entfernen einer eigenen Copperliste:

```
recall_old_copperlist:
         MOVE.L savecop,A5
         MOVE.L savecop2(A5)
         RTS
```

Label, die von den Copper-Routinen gebraucht werden:

```
gfxname: dc.b "graphics.library",0
even
savecop: dc.l 0
savecop2: dc.l 0
```

(Warnung der Redaktion: Sie sollten wissen was Sie tun, wenn Sie in dieser oder anderer Form direkt auf die Hardware des Amiga zugreifen.) *(Christian Zaenker/rs)*

Abfrage der Sondertasten in Assembler

```
cmp.b #$33 , $BFEC01
bne   loop
```

Diese Schleife wartet, bis die Commodore-Taste (A500) beziehungsweise die linke Amiga-Taste (A1000,A2000) gedrückt wird. Das kurze Programm vergleicht das Hardware-Register $BFEC01 mit dem Absolutwert $33. Dieser Wert entspricht dem Ereignis: »Commodore-Taste gedrückt« beziehungsweise »linke Amiga-Taste gedrückt«. Die folgende Tabelle gibt einige weitere Werte an, die bei Drücken bestimmter Tasten im Register $BFEC01 stehen.

Taste	Hex-Wert	Dez-Wert
<rightAmiga>	$31	49
<leftAmiga>	$33	51
<CTRL>	$39	57
<ALT>	$37	55
<rightShift>	$3d	61
<leftShift>	$3f	63

(Marco Borla/rs)

Profi-Tips für Programmierer

Tricks und Kniffe sind für effektives Programmieren unentbehrlich. Stöbern Sie in der Auswahl der besten Tricks zu C und Assembler, die wir aus den Jahrgängen 1987 und 1988 des AMIGA-Magazins zusammengestellt haben.

Direkt in die Bitmap

Für besonders schnelle Assembler-Programme ist es oft wünschenswert, Punkte auf dem Bildschirm direkt durch Adressierung der Bitplanes anzusprechen. Eine Möglichkeit hierzu bieten die zwei Unterroutinen »PSet« und »PClear«:

```
PClear:
      move.b #$b5, label+1
PSet:
      divu    #8, d0
      mulu   #40, d1
      add.w       d0, d1
      swap        d0
      move.b  #7,  d2
      sub.b   d0,  d2
label:
      bset    d2, (a5,d1)
      move.b #$f5, label+1
      rts
```

Die Unterprogramme sind mit dem Seka-Assembler geschrieben. Um einen Punkt zu setzen oder zu löschen, rufen Sie »PSet« beziehungsweise »PClear« auf. Übergeben Sie folgende Werte in den genannten Registern:
X-Koordinate des Punktes in d0
Y-Koordinate des Punktes in d1
Zeiger auf Bitplane in a5.

Die Startadressen der Bitplanes eines Screens sind leicht zu finden. Die Intuition-Funktion »OpenScreen« liefert den Zeiger auf die Screendaten. Addieren Sie zu diesem Wert #$C0. An der berechneten Stelle befindet sich die Adresse der ersten Plane. Der Pointer auf die zweite und alle weiteren Planes liegt jeweils 4 Byte höher.

Diese direkte Programmierung der Bitplanes ist etwas für Tüftler und birgt viele Fehlerquellen. Sie können Punkte auch durch einfache Angabe der Koordinaten mit Hilfe der Systemroutinen in der »graphics.library« setzen und löschen. In Basic steht Ihnen hierfür der Befehl PSET zur Verfügung. Allerdings sind diese Befehle ein wenig langsamer. *(David Czock/rs)*

Super-Hochauflösung

Überall steht geschrieben, der Amiga hätte eine maximale Bildschirmauflösung von 640 x 512 Punkten. In Wahrheit steht Ihnen eine weit höhere Auflösung zur Verfügung. Ein Screen kann bis zu 704 x 564 Punkte groß sein. Die 69376 zusätzlichen Punkte können Sie mit Hilfe der Grafikbefehle des Betriebssystems ohne weiteres nutzen. Besonders in C und Assembler ist dies recht einfach. Zwei Zahlen müssen geändert werden:

Um einen Screen zu öffnen, müssen Sie eine »NewScreen«-Struktur erstellen. Der dritte Wert dieser Tabelle gibt die Screenbreite an, der vierte die Höhe. Üblicherweise stehen dort, beispielsweise für einen hochauflösenden Screen im Interlace-Modus die Werte 640 und 512. Ersetzen Sie diese Werte durch 704 beziehungsweise 564.

```
NewScreen.Width=704;
NewScreen.Height=564;
```

Wenn Sie nun die Funktion »OpenScreen« anspringen, erscheint der übergroße Screen. Er ist aber nur zum Teil sichtbar. Um dem abzuhelfen, müssen Sie folgendes ändern:

Erstens sollten Sie das Bild des Monitors etwas verkleinern und neu zentrieren. Die Regler hierzu befinden sich im allgemeinen an der Rückseite des Monitors.

Zweitens müssen Sie das Bild mit Hilfe der »Preferences« nach oben links verschieben. Klicken Sie hierzu das große Rechteck in der Mitte des »Preference-Windows« an und bewegen das gesamte Fenster mit der Maus in die gewünschte Ecke. Für die genaue Einstellung ist ein wenig Experimentieren notwendig. Sie dürfen das Bild nicht zu weit verschieben, da sonst der obere Rand des Screens verschwindet. Wenn Sie den Monitor einmal eingestellt haben, können Sie in Ihren Programmen sofort die »Super-Hochauflösung« ausprobieren.

Der Effekt läßt sich im übrigen auch in allen anderen View-Modi ausnutzen, um mehr Punkte darzustellen. So stehen Ihnen zum Beispiel für einen LowRes-Screen 352 statt der üblichen 320 Punkte in der Horizontalen zur Verfügung.

Warum nicht gleich die volle Auflösung?

Die Einschränkung auf 640 x 512 Punkte erfolgt, weil nicht alle Monitore diese extreme Auflösung darstellen können (Overscanning). Besonders an den Rändern kommt es zu Unregelmäßigkeiten. Außerdem sind in den Grenzbereichen nicht alle Sprites sichtbar. *(Jürgen Brendel/rs)*

Zufallszahlen in Assembler

Für Grafiken, Spiele und mathematische Funktionen benötigen Programmierer häufig Zufallszahlen. Basic unterstützt diesen Wunsch durch das Statement RND(). In Assembler muß sich der Programmierer selber helfen. Er kann auf einige Hardwareregister zugreifen, die sich laufend ändern. Zum Beispiel ein Register, dessen Wert der Position des Rasterstrahls auf dem Bildschirm entspricht:

```
rnd:
 clr.l  d0             ; Register löschen
 move.w $dff006,d0     ; zufälligen Wert laden
 cmp.w   #0,d1         ; ist eine Obergrenze da?
 beq c
a:
 cmp.w d1,d0           ; Zufallswert zu groß?
 bge b
c:
 rts                   ; Rücksprung
b:
 sub.w d1,d0           ; Grenze abziehen
 bra a                 ; weiter vergleichen
```

Um die Routine aufzurufen, müssen Sie erst die größte erlaubte Zufallszahl in d1 übergeben. Andernfalls löschen Sie dieses Register. Füllen Sie doch einmal ein Fenster mit Zufalls-Punkten:

```
start:
ExecBase:        EQU   4
OpenLib:         EQU -408
CloseLibrary:    EQU -414
movem.l d0-d7/a0-a6,-(sp)    ; Register retten
 move.l ExecBase,a6          ; Grafics.lib öffnen
 lea Graphicsname,a1
 jsr OpenLib(a6)
 move.l d0,GFX_Base
 lea Intname,a1              ; Intuition.lib öffnen
 jsr OpenLib(a6)
 move.l d0,Int_Base
  move.l #1000,d5            ; Zahl der Punkte
  move.l GFX_Base,a6
d:
   move.w #200,d1            ; y-max
   jsr rnd
   move.w  d0,d3             ; retten
   move.w #600,d1            ; x-max
   jsr rnd
   move.w d3,d1
   move.l Int_Base,a1       ; Zeiger auf aktuelles
   move.l $34(a1),a1        ; Fenster ermitteln
   move.l 50(a1),a1         ; Zeiger auf Rastport
   jsr -324(a6)             ; Draw Pixel
  dbra d5,d                 ; nächster Punkt
 move.l ExecBase,a6         ; alles wieder schließen
 move.l GFX_Base,a1
 jsr CloseLibrary(a6)
 move.l Int_Base,a1
 jsr CloseLibrary(a6)
 movem.l (sp)+,d0-d7/a0-a6
 rts                         ; zurück
  EVEN
Graphicsname: dc.b 'graphics.library',0
  EVEN
Intname:      dc.b 'intuition.library',0
  EVEN
GFX_Base: dc.l 0
Int_Base: dc.l 0
```

Sie können beide Listings in der angegebenen Fassung sowohl mit dem Seka- als auch dem auf der Fish-Disk 110 befindlichen »A68k«-Assembler übersetzen.

Da die Zufallsroutine ein sich kontinuierlich veränderndes Register verwendet, sind die nacheinander abgerufenen Zahlen nicht vollkommen unabhängig. Besser wäre es, die Werte mehrerer Hardwareregister zu kombinieren, um Zufallszahlen zu erzeugen. Zufallszahlen lassen sich auch mathematisch berechnen. Vielleicht haben Sie schon einen passenden Algorithmus hierzu entwickelt? *(Gerhard Mattner/rs)*

Schnelle Schleifen in C

Bei Schleifen lohnt es sich, Zeit zu sparen:

```
main ()
int x
for ( x=0; x < 5000000 ; x++)
```

Diese Schleife mit 5 Millionen Durchläufen benötigt etwa 46 Sekunden, wenn Sie das Programm von der RAM-Disk starten. Wesentlich schneller wird die Schleife durchlaufen, wenn Sie Registervariablen verwenden:

```
main()
register int x;
for ( x=0; x < 5000000; x++)
```

Dieser »Loop« benötigt nur 23 Sekunden. Eine vergleichbare Assembler-Schleife ist noch schneller:

```
Start:
 move.l #5000000,d2
loop:
 dbra d1, loop
rts
```

Eine solche Schleife benötigt nur noch 7 Sekunden.

(Andreas Moshammer/rs)

Bildschirmsplitting

Diese kleine Farbdemonstration splittet den Bildschirm in zwei Hälften. Das Programm demonstriert die Abfrage der Position des Rasterstrahls mit Hilfe der Hardware-Register.

```
 execbase = 4
 forbid = -132
 permid = -138
start:
 movem.l d0-d2/a1/a2/a6,-(SP)
 lea $dff004,a1      ; u.a. Y-POS des Rasterstrahls
 lea $dff180,a2      ; Farbregister
 move.l execbase,a6
 jsr forbid(a6)      ; Multitasking aus
zeilenloop:
 move.l #$2b00,d0    ; >>2b<< = Zeile für Split
                     ; >>1b<< = Minimum
Farbloop:
 move.w #$0000,d1    ;erste Farbe
wait:
 move.l (a1),d2      ;
 and.l #$0001ff00,d2; y-Pos. in d2
 cmp.l d0,d2         ; Zeile erreicht ?
 bne wait
 move.w d1,(a2)      ; neue Farbe ins Register
wait1:
 move.l (a1),d2
 and.l #$0001ff00,d2
 cmp.l d0,d2
 beq wait1           ; nächste Zeile abwarten
 cmp.b #$bc,$bfe001
 beq done            ;linke Maustaste gedrückt ?
 add.w #$100,d1      ;neue Farbe
 cmp.w #$1000,d1     ;Farbe wechselt laufend
 blt wait
 add.l #$0100,d0     ;Splitzeile +1
 cmp.l #$13900,d0    ;größer Maximum?
 blt farbloop
 bge zeilenloop
done:
 move.l execbase , a6
 jsr permid(a6)
 movem.l (sp)+,d0-d2/a1/a2/a6
 rts
```

Das Programm ist mit dem Seka-Assembler geschrieben. Nach dem Start wird zunächst jeder weitere Task gesperrt. Danach verändert das Programm ab einer bestimmten Bildschirmzeile die Hintergrundfarbe. Zur besseren Demonstration wird diese Farbe laufend geändert. Nach Durchlaufen der Farbskala erhöht das Programm die Zeile, in der der Bildschirm gesplittet werden soll. Sie können den Programmablauf jederzeit durch einen Druck auf die linke Maustaste stoppen. Ein Nachteil des Programms ist, daß der Prozessor auf eine bestimmte Rasterposition wartet und nichts anderes erledigen kann. Noch besser wäre es daher, die Farbe des Hintergrunds mit Hilfe des Coppers ab einer bestimmten Position zu verändern, aber das ist eine andere Geschichte.

(Klaus Kuphal/rs)

Es geht auch ohne Maus

Gehören Sie zu den Anwendern, die die Maus auch mal beiseite legen und nur mit dem CLI arbeiten? Insbesondere C- und Assemblerprogrammierer benötigen keine Workbench-Umgebung. Dann werden Sie sicherlich eine Diskette anlegen, bei der das AmigaDOS-Fenster nach dem Booten gar nicht erst geschlossen wird. Am einfachsten geht dies, indem Sie den Befehl LOADWB in der »Startup-Sequence« entfernen. Dabei entsteht das Problem, daß das Fenster nach dem Start mit der Maus auf die volle Größe gebracht werden muß, um die PAL-Auflösung zu nutzen. Doch wo ist die Maus? Wieder müssen Sie dieselbe aus der Schreibtischschublade hervorkramen. Aber halt, es geht auch einfacher. Dieses C-Programm, in die »Startup-Sequence« eingebunden, schafft Abhilfe:

```
#include <exec/types.h>
#include <intuition/intuition.h>
struct IntuitionBase *IntuitionBase;
main()
{
IntuitionBase = (struct IntuitionBase*)
     OpenLibrary ("intuition.library",0);
if (IntuitionBase == 0) exit (0);
  SizeWindow (IntuitionBase -> ActiveWindow, 0,
  56);
  CloseLibrary (IntuitionBase);
}
```

In den ersten beiden Zeilen werden die benötigten Include-Files eingebunden. Der Zeiger auf die IntuitionBase wird eingerichtet und in main() die Intuition-Bibliothek geöffnet. Die Funktion »SizeWindow« verändert die Größe eines Windows:

```
SizeWindow (Zeiger auf Fensterstruktur, dy, dx)
```

In diesem Fall wird das aktuelle Fenster, also das CLI-Window um 56 Pixel in der Vertikalen vergrößert. Zum Schluß wird die Library geschlossen und zurück zum CLI gesprungen. Eine einfache, aber wirkungsvolle Sache.

(Dirk Brammerts/rs)

Ein- und Ausblenden in C

Die folgende Funktion demonstriert Ihnen die Technik, Texte und Grafiken einzublenden:

```
einblende() {
char i;
for (i=0;i<=15;i++)  {
  SetRGB4(window->RPort,1,i,i,i);
  Delay (2);          }
}
```

Um mit dieser Funktion ein Einblenden zu erreichen, setzen Sie zunächst in einem Fenster mit Namen »window« die Farben 0 und 1 auf Schwarz. Verwenden Sie hierzu die SetRGB4-Funktion aus der »graphics.library«. Tragen Sie statt der Variablen i den Wert Null ein. Als nächstes schreiben Sie Ihren Text mit der Farbe 1 in das Fenster. Natürlich erkennen Sie noch nichts. Alles ist »schwarz auf schwarz«. Erst durch den Aufruf von »einblende()« wird der Text langsam sichtbar. Mit der Funktion »Delay()« aus der »dos.library« bestimmen Sie die Verzögerung.

Die Routine zum Einblenden läßt sich vielfältig manipulieren. Ersetzen Sie beispielsweise im Schleifenkopf die Zeile:

```
for (i=15;i>0;i--)
```

Nun haben Sie die Funktion zum Ausblenden eines Textes programmiert. Die angegebenen Libraries müssen Sie natürlich vorher öffnen, sonst klappt nichts.

(Peter Hug/rs)

Gute Reaktion im CLI

Manchmal möchte man die »Startup-Sequence« anhalten und auf eine Reaktion des Benutzers warten lassen. Es bietet sich an, einen CLI-Befehl zu schreiben. In C ist dies relativ einfach: Sie können das Makro »getchar« aus »stdio.h« verwenden:

```
#include <stdio.h>
 main()
{
getchar();
}
```

Das ist einfach. Das kleine Programm wartet auf das Betätigen einer Taste. Noch professioneller sieht ein Programm aus, das auf den Druck auf eine Maustaste wartet. Dafür gibt es noch kein Makro. Doch Sie können dem Programm über IDCMP eine Meldung schicken, wenn die Maustaste gedrückt wird. Hierzu brauchen Sie den Zeiger auf das Fenster, in welchem Ihr Programm läuft. Glückli-

cherweise kann man den Pointer (Zeiger) auf das aktuelle CLI-Fenster über Intuition-Base abfragen:

```
/* WaitMouse.c - Wartet auf Linke Maustaste */
#include <intuition/intuitionBase.h>
struct IntuitionBase *Intuitionbase;
struct Window *WindowPtr;
main()
{
IntuitionBase =(struct IntuitionBase*)
        OpenLibrary("intuition.library",0);
WindowPtr = IntuitionBase->ActiveWindow;
ModifyIDCMP(WindowPtr,MOUSEBUTTONS);
Wait(1<<WindowPtr->UserPort->mp_SigBit);
ReplyMsg(GetMsg(WindowPtr->UserPort));
ModifyIDCMP(WindowPtr,0);
CloseLibrary(IntuitionBase);
}
```

In compilierter Form (+L Option) kann man das Programm wunderschön als CLI-Befehl verwenden und in eigene Start-Sequenzen einbauen. *(Rade Sotonica/rs)*

Film ab: Bitte nicht stören

Bei der Programmierung von animierter Grafik in C kann es vorkommen, daß die Grafik stark ruckelt, wenn man bei laufendem Programm eine Diskette in das Laufwerk legt, die Mausknöpfe drückt oder die Tastatur benutzt.

Durch eine Erhöhung der Task-Priorität läßt sich dies vermeiden. Das folgende Beispiel zeigt wie es gemacht wird:

```
#include <exec/types.h>
#include <exec/tasks.h>

struct Task *Task

main ()
{
 Task = FindTask(0) ;
 /* aktuellen Task ermitteln*/
 SetTaskPri (Task,127); /* Priorität erhöhen*/
...
 /* Hier Programm einfügen */
...
 SetTaskPri (Task,0) ; /*Priorität zurücksetzen*/
}
```

Nach Ermitteln des aktuellen Tasks durch die Funktion »FindTask«, wird dessen Priorität auf 127 gesetzt. Am Ende muß aber wieder der alte Zustand hergestellt werden. *(Stefan Brabec/rs)*

Start von der Workbench

Die meisten Assembler-Programmierer werden wohl schon einmal gemerkt haben, daß ein Maschinenprogramm beim Start von der Workbench (per Mausklick) erst einmal eine Message (Botschaft) entgegennehmen muß. Wie dies gemacht wird, steht in den meisten Büchern, die sich mit Assembler-Programmierung beschäftigen. Was oft zu kurz kommt, ist die Tatsache, daß die Botschaft vom Programm erwidert werden muß. Vergißt man die Antwort, wartet die Workbench vergeblich auf den Rückruf; die Message wird zur »Speicherleiche«. Man erkennt diesen Zustand daran, daß das Icon der Programm-Diskette nicht verschwindet, wenn sie aus dem Laufwerk genommen wird.

Ein korrektes Beispiel für die Annahme einer Botschaft zeigt das Listing für den Seka-Assembler:

```
execbase = 4
Forbid   = -132
FindTask = -294
WaitPort = -384
GetMsg   = -372
ReplyMsg = -378
Start:
 move.l execbase, a6
 move.l a7, Stackpointer ; Stackzeiger retten
 clr.l  Message          ; Messagepointer auf 0
 sub.l  a1,a1            ; a1 = 0
 jsr    FindTask(a6)     ; Zeiger auf eigenen Task
 move.l d0,a4
 tst.l  $ac(a4)          ; Test auf Start vom CLI
 bne    Main             ; ja, dann Sprung
 lea    $5c(a4),a0       ; Zeiger auf MessagePort
 jsr    WaitPort(a6)     ; auf Botschaft warten
 jsr    GetMsg(a6)       ; und entgegennehmen
 move.l d0,Message       ; Message retten
Main:
....
....                     ; eigentliches Programm
....
Ende:
 tst.l  Message          ; Message holen
 beq.s  NoMessage
 move.l execbase,a6
 jsr    Forbid(a6)       ; Multitasking aus
 move.l Message,a1
 jsr    ReplyMsg(a6)     ; Antworten
NoMessage:
 move.l Stackponter,a7   ; Stackpointer herstellen
 clr.l  d0               ; kein Fehler
data
Stackpointer: dc.l 0
Message:      dc.l 0
end
```

Das Retten des Stackpointers und das Aufrufen der Funktion Forbid ist absolut notwendig; sonst wird der Guru aktiv. Mit den gezeigten Ergänzungen ist ein Programm sowohl vom CLI als auch von der Workbench lauffähig. Sie können das Programm auch an jede Stelle (auch in Unterprogrammen) mit »bra ende« verlassen. Der Zeiger auf den Stack wird am Ende ohnehin »restauriert«. *(Martin Gräfe/rs)*

Joystickabfrage in Assembler

Die Verwendung von Assembler ist eine Alternative zu Basic oder C, um schnelle Programme zu erzeugen. Um von einem Maschinenprogramm den Joystick abzufragen, gibt es zwei Verfahren:

1. Sie können das »Gameport-Device« verwenden. Der Haken an der Sache: Ihr Programm wird aufgebläht und langsam.
2. Als Alternative bleibt, den Joystickport direkt abzufragen. Greifen Sie mittels weniger Assemblerbefehle direkt auf die Hardwareregister zu. Das folgende Programm kehrt nach seinem Aufruf mit einem Wert in d0 für den Joystick in Port 0 zurück. Den entsprechenden Wert für Port 1 finden Sie in d1:

```
 JOY0DAT =$DFF00A
 JOY1DAT =$DFF00C
 JOYTEST =$DFF036
LeseJoystick:
 clr.w JOYTEST      ; Löschen der Mausregister
 move.w JOY0DAT,d0 ; JoyPort 0 lesen
 move.w Joy1DAT,d1 ; JoyPort 1 lesen
 rts
```

In den Registern d0 und d1 stehen je nach Stellung des angeschlossenen Joysticks diese Werte:

$0200 links
$0300 links oben
$0201 links unten
$0100 oben
$0000 Mitte
$0001 unten
$0102 rechts oben
$0002 rechts
$0003 rechts unten

Einfacher und schneller geht es nicht. *(Jörg Schmidt/rs)*

Der MC 68010 im Amiga

Diejenigen Leser, die ihren Amiga mit dem MC 68010 aufgerüstet haben (s. Ausgabe 8/9 1987 des AMIGA-Magazins), mußten feststellen, daß einige wenige Programme hartnäckig ihren Dienst verweigern. Der Grund liegt in der Betriebssystemroutine »User State«, die nicht berücksichtigt, daß der 68010 bei einer »Exception 4« vier Worte auf den Supervisor-Stack ablegt. Abhilfe schafft hier ein kleines Assembler-Programm namens »NewUserState«, das einfach eine neue UserState-Routine im Speicher installiert und sinnvollerweise gleich mit der Startup-Sequence aufgerufen wird. Genau wie das Original im ROM benötigt die neue Routine beim Aufruf als einzigen Parameter die Adresse des Supervisor-Stacks im Datenregister D0.

```
ExecBase:      EQU 4
AllocMem:      EQU -198
UserState:     EQU -156
InstallPatch:
MOVE.L         ExecBase,A6
MOVE.L         #Patchlänge,D0
CLR.L          D1
JSR            AllocMem(A6)
TST.L          D0
BNE.S          Okay
MOVEQ          #100,D0
RTS
Okay:
MOVE.L         D0,A0
MOVE.L         D0,A2
MOVE.L         #Patchlänge,D0
LEA            NewUserState,A1
Loop:
MOVE.B         (A1)+,(A0)+
DBF            D0,Loop
MOVE.L         A2,2+Userstate(A6)
CLR.L          D0
RTS
; Hier nun die eigentliche Routine,
; die in den Speicher kopiert wird.
NewUserState:
OR.W           #$2000,SR
MOVE.W         SR,D1
MOVE.L         A7,USP
MOVE.L         D0,A7
BCLR           #$000D,D1
MOVE.W         D1,SR
RTS
PrEnde:
Patchlänge: EQU PrEnde-NewUserState
```

(Michael Paar/rs)

Aus die Maus

Schalten Sie den Mauszeiger aus, wenn er Sie stört: Wer nur mit dem CLI arbeitet, fühlt sich eventuell häufig durch den Zeiger der Maus belästigt. Immer erscheint er dort, wo etwas Wichtiges steht und dann ist die Maus gerade nicht zur Hand. Mit zwei kleinen Programmen können Sie die Maus an- und ausschalten:

```
/* MouseOff */
#include <exec/types.h>
#include <intuition/intuitionbase.h>
#include <intuition/intuition.h>
#include <functions.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

#define LIBRARY_VERSION 0
#define POINTER_HEIGHT 15
UWORD UnvisibleImage [POINTERHEIGHT * 2 + 4]=
 {
 0, 0,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
 0, 0
 };
main()
 {
 if ((IntuitionBase = (struct IntuitionBase *)
     OpenLibrary ("intuition.library",
LIBRARY_VERSION)) == NULL)
 exit(10);
 if ((GfxBase = (struct GfxBase *)
     OpenLibrary ("graphics.library",
LIBRARY_VERSION)) == NULL)
 exit(10);

 SetPointer(IntuitionBase->
 ActiveWindow, &UnvisibleImage[0],15,15,-8,-7);
 printf("Weg isser....\ n\ n");
CloseLibrary(GfxBase);
CloseLibrary(IntuitionBase);
}
```

Wenn Sie die Maus wieder sehen möchten, verwenden Sie das zweite Programm:

```
/* MouseOn */
#include <exec/types.h>
#include <intuition/intuitionbase.h>
#include <intuition/intuition.h>
#include <functions.h>
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
#define LIBRARY_VERSION 0
#define POINTER_HEIGHT 15
main()
 {
 if ((IntuitionBase = (struct IntuitionBase *)
     OpenLibrary ("intuition.library",
LIBRARY_VERSION)) == NULL)
 exit(10);
 if ((GfxBase = (struct GfxBase *)
     OpenLibrary ("graphics.library",
LIBRARY_VERSION)) == NULL)
 exit(10);
 ClearPointer (IntuitionBase->ActiveWindow);
 printf("Wieder da\ n\ n");
CloseLibrary(GfxBase);
CloseLibrary(IntuitionBase);
}
```

Beide Programme sind mit dem Aztec-C-3.40a geschrieben. Die Aufrufe zum Compilieren lauten:

```
cc MouseOff +l
ln MouseOff.o -lc32 +CD
```

»MouseOn.c« übersetzen Sie analog. Die Option +CD ist nicht unbedingt erforderlich. Und wenn Ihre Maus jetzt noch einmal stört, wissen Sie, was Sie tun müssen. *(Stefan Sperling/rs)*

Gute Vorarbeit von Aztec-C

Bei dem C-Compiler von Aztec handelt es sich schon um einen schnellen Compiler. Nur wenn der Programmierer mit vielen Include-Dateien arbeitet, dauert die Übersetzung eines Programms recht lange. Dies macht sich vor allem bei der Entwicklung eines Programms bemerkbar, wenn der Programmierer den Compiler häufig einsetzt. Bei jeder kleinsten Änderung im Quell-Text muß ein Programm neu compiliert werden und jedesmal müssen die Include-Dateien neu geladen werden. Wer viel Speicher besitzt, kann diese Dateien zumindest in der RAM-Disk speichern. Aber an den

Include-Dateien ändert sich in der Regel bei der Software-Entwicklung nichts. Müssen diese Dateien immer von neuem übersetzt werden? Könnte man nicht Zeit sparen, wenn die Dateien quasi vorcompiliert vorlägen. Genau diese Option wird von Aztec-C bereits unterstützt. Die Lösung heißt »pre-compiling«.

Dabei wird ein File compiliert, aber noch kein Assemblercode erzeugt. Statt dessen liefert der Compiler quasi eine Vorstufe des übersetzten Quellcodes und speichert die Informationen wie etwa Symboltabellen auf Diskette.

Beim endgültigen Compilieren wird dieser Code wieder geladen. Die Inlucde-Anweisungen im Quellprogramm können dann entfallen.

Und wie funktioniert das in der Praxis? Schauen wir uns ein Beispiel an. Man erstellt zunächst ein Programm-Segment, das nur aus Include-Anweisungen besteht:

```
INCLUDE <stdio.h>
INCLUDE <intuitionbase.h>
INCLUDE <exec/execbase.h>
....
```

Nun rufen Sie den Compiler auf:

```
cc Name [Optionen] +hINCLUDE.PRE
```

— Name kennzeichnet unser Programm, das die Include-Aufrufe enthält.
— Die Optionen sind genauso zu setzen, wie beim normalen Compilieren.
— INCLUDE.PRE schließlich ist der Name des precompilierten Files. Achten Sie darauf, daß zwischen +h und INCLUDE.PRE keine Leerzeichen auftauchen.

Um die präparierte Datei benutzen zu können, ist der folgende Aufruf notwendig:

```
cc programmname [Optionen] +iINCLUDE.PRE
```

Bevor Sie diese neue Technik ausprobieren — Sie werden begeistert sein — noch ein Hinweis: Im normalen Programmcode dürfen jetzt natürlich nicht mehr die Include-Dateien aufgerufen werden, die bereits übersetzt wurden. Ansonsten tritt ein Fehler auf (»Label declared twice«). Andere Includes dürfen natürlich verwendet werden. Aber warum noch Includes im Quell-Code verwenden? Es geht doch jetzt auch mit vorcompilierten Dateien.

(Michael Friedrich/rs)

Panorama-Fenster oder Bullauge

Es existieren schon viele Programme, die dafür sorgen, daß das Amiga-DOS-Fenster auf die volle Größe des Bildschirms vergrößert. Die Variante »size« ist etwas Besonderes:

```
#include "intuition/intuitionbase.h"
struct IntuitionBase *IntuitionBase;
VOID main(argc, argv)
SHORT argc;
TEXT *argv[];
{
 struct Window *Window;
 if((argc!=2)||((*argv[1]!='b')&&(*argv[1]!='s')))
 {
  printf("Usage: %s [s][b]\n", argv[0]);
  exit(FALSE);
 }
 if(!(IntuitionBase=OpenLibrary("intuition.library", 0)))
 {
  printf("Can't open Intuition\n");
  exit(FALSE);
 }
 Window=IntuitionBase->ActiveWindow;
 if(Window!=NULL)
 {
 if(*argv[1]=='b')
 {
 MoveWindow(Window, -Window->LeftEdge, -Window->TopEdge);
 SizeWindow(Window, IntuitionBase->ActiveScreen->
             Width-Window->Width, IntuitionBase->
             ActiveScreen->Height-Window->Height);
 WindowToFront(Window);
 }
 else if(*argv[1]=='s')
 {
 SizeWindow(Window, -Window->Width+85, -Window->Height+23);
 MoveWindow(Window, IntuitionBase->ActiveScreen->
          Width-85, IntuitionBase->ActiveScreen->
                Height-23); WindowToBack(Window);
 }
 }
 CloseLibrary(IntuitionBase);
}
```

Wenn Sie C-Programme nach dem Compilieren ins Verzeichnis »c« Ihrer Workbench kopieren, können Sie das DOS-Fenster jederzeit auf die volle Größe bringen. Rufen Sie lediglich »size b« auf. Der Befehl kann auch in der »Startup-Sequence« stehen. Und was ist das Besondere an »size«? Wenn Sie »size s« verwenden, verkleinert der Amiga das DOS-Window und schiebt es rechts unten in die Ecke.

(Peter Fettke/rs)

Metacomco unter Kontrolle

Gibt man während des Assemblierens mit dem Assembler von Metacomco (V11.00) <CTRL F> ein, wird der aktuelle Status im folgenden Format ausgegeben:

```
*** Pass x   Line y   Errors z
```

Eine ausgezeichnete Möglichkeit, um den Stand der Übersetzung zu verfolgen.

(Stefan Riege/rs)

Schwupps, weg ist es

Auch Sie kennen bestimmt den System-Requester »TASK-HELD«. Wenn Sie aber noch einen weitereren CLI-Prozeß laufen haben, können Sie meistens noch sehr lange weiterarbeiten, ohne daß der Rechner ganz abstürzt. Doch es stört, wenn dann das zweite oder dritte dieser Fenster auf dem Bildschirm steht. Mit dem Programm »whap« können Sie die störenden Fenster eliminieren. Rufen Sie einfach Whap auf und klicken Sie das Fenster an, das verschwinden soll:

```
#include <intuition/intuitionbase.h>
struct IntuitionBase *IntuitionBase;
VOID main()
{
 if(!(IntuitionBase=(struct IntuitionBase*)
    OpenLibrary("intuition.library", LIBRARY_VERSION)))
 {
  printf("Can't open Intuition!\n");
  exit(FALSE);
 }
 Delay(300);
 if(IntuitionBase->ActiveWindow)
       CloseWindow(IntuitionBase->ActiveWindow);
 CloseLibrary(IntuitionBase);
}
```

Zur Funktionsweise: Nachdem das Programm die »intuition.library« geöffnet hat, wartet es etwa sechs Sekunden — danach schließt »whap« einfach das aktive Window.

(Peter Fettke/rs)

Verbessertes CLS in C

Das Bildschirmlöschen geht in C auch mit »puts("\f");«. Das ergibt, zumindest beim Aztec-C, kompakteren Code und damit kürzere Ladezeiten eines CLS-Programms. Mit printf() sollte jeder schreiben:

```
printf("\f");
```

statt

```
printf("\%c",12)
```

Der Code 12 ist zwar beim Amiga und allen MS-DOS-Computern ein Seitenvorschub, aber nicht auf allen Computern. Im Interesse der Portabilität und der Verständlichkeit sollte jeder C-Programmierer die Standardzeichen verwenden.

(Tassilo Schinhammer/rs)

Was tun bei »Task held«?

Wer einen schwerwiegenden Programmierfehler gemacht und den Amiga dadurch zum Absturz gebracht hat, kennt die Meldung »Task held, finish all Disc-activity«. Was viele aber nicht wissen: Diese Meldung besagt, daß lediglich dieser eine Task angehalten wurde. Wenn Sie zu diesem Zeitpunkt noch einen weiteren Task laufen haben, zum Beispiel ein zweites CLI, können Sie mit diesem problemlos weiterarbeiten. Vor allem Einsteiger schalten bei dieser Meldung nämlich den Computer aus und wieder ein, da sie meinen, er wäre vollständig abgestürzt. *(Teut Weidemann/rs)*

Der Profi-Tip: Eingebauter Debugger beim Amiga

Wußten Sie schon, daß im Amiga-Betriebssystem (im Kickstart-ROM) ein Debugger eingebaut ist? Er heißt »ROM-Wack« und ist ein Subset des Debuggers, der auf der Makroassembler-Disk aus dem Entwicklerpaket unter dem Namen WACK gespeichert ist. ROM-Wack wird bei einem »Guru Meditation Error« angesprungen, wenn am RS232-Port des Amiga ein Terminal angeschlossen ist (vielleicht sogar ein alter C 64 mit Terminalprogramm oder ein unwürdiger MS-DOS-Clone). Über dieses Terminal kann dann im Speicher des Amiga auf Fehlersuche gegangen werden.

ROM-Wack läßt sich auch vom CLI aus aufrufen. Ist die Workbench noch nicht geladen (also <CTRL D> beim Booten), kann mit dem Befehl »LOADWB -DEBUG« das ROM-Wack als Bestandteil der Workbench geladen werden. Probieren Sie jetzt bei gedrücktem rechten Mausknopf auf der Workbench-Menüzeile entlang nach rechts zu fahren. Ganz rechts erscheint plötzlich ein vorher verstecktes Menü. Damit können Sie das über die serielle Schnittstelle laufende ROM-Wack aktivieren und steuern, auch wenn der Amiga bereits keinen Ton mehr von sich gibt (sprich: rettungslos abgestürzt ist). Eine ausführliche Anleitung zum ROM-Wack findet sich im (englischsprachigen) »Amiga-DOS Developers Manual«. *(Manfred Kohlen/rs)*

Bildschirmscrollen in »C«

Mit dem Befehl »ScrollRaster(rp,x,y,xmin,ymin,xmax,ymax)« läßt sich in »C« der Bildschirm auf einfache Weise scrollen. Dabei ist »rp« ein Pointer auf ein Rasterport, x und y sind die Anzahl der Punkte, die in x- oder y-Richtung gescrollt werden sollen und die restlichen vier Parameter die Koordinaten des zu scrollenden Bildschirmausschnittes. *(Cord-Henning Scholz/rs)*

Laufwerk: eine brennende Sache

Nachdem im AMIGA-Magazin 2/88, Seite 93, das Ein- und Ausschalten der Power-LED gezeigt wurde, ist nun die Leuchtdiode des Laufwerks dran. Das Aufleuchten erreichen Assemblerprogrammierer mit diesen Befehlen:

```
LED:  EQU $BFD100
      move.b #127,LED
      move.b #119, LED
      move.b #0, LED +512
```

Auch wer in Basic programmiert, kann die LED aufleuchten lassen:

```
LED = 12570880&
POKE LED, 127
POKE LED, 119
POKE LED + 512,0
```

Um die LED wieder auszuschalten, braucht man lediglich die letzte 0 mit dem Wert 255 auszutauschen. Das kleine Programm kann jeder gut brauchen, der zum Beispiel dem Anwender zeigen möchte, daß er die Finger vom Laufwerk lassen soll. So verhindern Sie, daß jemand eine Diskette zu früh aus dem Laufwerk nimmt. Entwarnung ist erst, wenn die Lampe erlischt.

(Armin Hegglin/rs)

Reservieren bestimmter Speicherbereiche

Die EXEC-Funktion »AllocAbs« ist sehr nützlich, wenn bestimmte Speicherbereiche reserviert werden sollen, um dort zum Beispiel eine Grafik abzulegen. Diese Funktion teilt dem Betriebssystem mit, daß der gewünschte Bereich im Speicher belegt ist. Schreibt man seine Daten in einen nicht reservierten Bereich, kann es passieren, daß das Betriebssystem diesen neu belegt. Mit dem folgenden Assembler-Programm wird zum Beispiel ein Speicherblock von $50000 bis $60000 reserviert:

```
BelegeAbs:
  move.l     $04,a6     ; AbsExecBase
  move.l  #$50000,a1    ; Startadresse
  move.l  #$10000,d0    ; Anzahl der Bytes
  jsr        -204(a6)   ; AllocAbs
```

(Christian Seiler/as)

Assemblen Sie mit

Assembler-Programmierer schätzen ihre Lieblingssprache, da sie den direkten Zugriff auf die Hardware erlaubt:

Das CIA-A-Register $BFE001 enthält nützliche Informationen. Die wichtigsten Bits dieses Registers und deren Funktionen sind:

Bit	Funktion
1	wenn = 1 Power LED dunkel
6	wenn = 0 dann Feuerknopf an Port 0 gedrückt
7	wenn = 0 dann Feuerknopf an Port 1 gedrückt

Die anderen Bits dienen zur Steuerung des Laufwerks. Die einzelnen Bits können mit dem Befehl:

```
BTST  #n,BFE001
```

abgefragt werden. Ist das Zero-Flag (Z) nach dem Test gesetzt, war das entsprechende Bit gelöscht.

Beim Amiga muß bekanntlich aufgrund der Multitaskingfähigkeit ein Speicherbereich vor der Benutzung reserviert werden. Hierzu dient die Funktion »AllocMem«. Nun ist es aber so, daß man bei der Verwendung der Custom-Chips unbedingt Chip-Memory benötigt. Für andere Anwendungen wird dann aber Speicher gebraucht, der im Fast-Memory liegen soll. Wie kann man festlegen, welcher Speicher reserviert wird? Der Funktion AllocMem in der Exec-Library kann neben der Größe des benötigten Speicherblocks (d0) auch noch ein Wort mit Bedingungen übergeben werden (requirements in d1):

Wert	Anforderung
2	unbedingt Chip-Memory
4	Fast-Memory
1	reservierter Speicher darf nicht verschoben werden (bei der jetzigen Version kommt dies nicht vor)
$20000	größten freien Speicherblock belegen
$10000	reservierten Speicher löschen

Es können auch mehrere Bedingungen übergeben werden. Ist die Bedingung = 0, wird wenn möglich Fast-RAM reserviert.

Bei manchen Programmen ist es notwendig, Routinen des Betriebssystems direkt anzuspringen (mit JSR FCXXXXX). Allgemein gesagt, ist es manchmal erforderlich, zu wissen, welche Kickstart-Version vorliegt. Es ist sinnvoll, in einem solchen Fall am Anfang des Programms zu testen, ob die richtige Version vorliegt. Eine Abfrage in Assembler sieht wie folgt aus:

```
LEA $FC0000
CMP.l #" 33.",$1C(a0)
BNE Fehler
CMP.l #"192 ",$20(a0)
BNE Fehler
```

Dieses kleine Programm prüft auf die Version 33.192. Sie können die Routine entsprechend Ihren eigenen Wünschen anpassen. *(Klaus Wenger/rs)*

Checksummer

Dieses Programm ist unentbehrlich beim Abtippen unserer Listings. Es hilft, Tippfehler zu vermeiden und spart viel Zeit.

Ein längeres Listing ohne Fehler abzutippen ist (fast) unmöglich. Aus diesem Grund haben wir in Ausgabe 3/88 des AMIGA-Magazins eine Eingabehilfe — den Checksummer »Checkie 42« veröffentlicht. Die hier vorgestellte Version 1.1 enthält erweiterte Funktionen und bietet mehr Komfort. Damit möglichst viele unserer Leser dieses Programm auch tatsächlich anwenden, haben wir es möglichst kurz gehalten und in einer Sprache programmiert, die alle Abtipper besitzen: Amiga-Basic.

Die Form der Listings

Die Listingzeilen bestehen aus einer bis zu vierstelligen Zeilennummer, der zwei- beziehungsweise dreistelligen Prüfsumme und der eigentlichen Programmzeile. Beispiel:

```
10 TTO print "Hallo!"
│   └──Prüfcode
└──────Zeilennummer
```

Nach einer Leerstelle im Anschluß an die Zeilennummer stehen bis zu drei Zeichen Prüfcode. Die einzelnen Zeichen können sein eine Ziffer (»0« bis »9«), ein kleiner Buchstabe (»a« bis »z«) oder ein Großbuchstabe (»A« bis »Z«). Die ersten beiden Zeichen der Prüfsumme sind der eigentliche Prüfcode. Im dritten Zeichen ist die Spaltenposition der ersten »Nicht-Leerstelle« verschlüsselt. Das ist für diejenigen Anwender interessant, welche die Struktur des Listings, also die Einrückungen durch Leerzeichen, übernehmen wollen. Ist dies nicht Ihre Absicht, können Sie die Eingabe der Checksumme schon nach den ersten beiden Zeichen mit <Return> abschließen. Bei Checkie 42 muß die Groß- und Kleinschreibung so wie im Listing abgedruckt übernommen werden.

> **Eingabehinweis:**
> Geben Sie »Checkie 42« (Version 1.1) bitte mit Checkie 42 aus der AMIGA-Ausgabe 3/88 oder 12/87 ein. Sollten Sie die alte Eingabehilfe nicht besitzen, so tippen Sie das Listing für die Version 1.1 im normalen Basic-Editor ohne Prüfsummen und Zeilennummern ab.

Der Umgang mit Checkie 42

Nach dem Start fragt das Programm nach einem Dateinamen. Unter dem angegebenen Namen speichert Checkie 42 die eingegebenen Listingzeilen ab. Existiert bereits eine Datei mit diesem Namen auf der Diskette, so haben Sie mit der Abfrage »Nur Checksummer ausgeben?« zwei Möglichkeiten:

<j> Ausgabe der Datei mit Checksumme auf den Bildschirm oder den Drucker.
<n> Einlesen der Programmzeilen aus der vorhandenen Datei und Eingabe der Checksumme mit der Tastatur.

Beide Alternativen sind gedacht für Anwender, die ein Listing nicht mit dem Zeileneditor des Checkie, sondern mit einem schnelleren und/oder komfortableren Editor ihrer Wahl — zum Beispiel dem Editor von Amiga-Basic (mit »..",a« speichern) erfaßt haben.

Checkie 42 errechnet nach der Eingabe <j> die Prüfsummen Ihres Textes und Sie können diese dann mit dem Listing im AMIGA-Magazin vergleichen. Bei der Ausgabe auf den Bildschirm schreibt das Programm die Programmzeilen inklusive Checksummen zusätzlich in eine Datei auf Diskette mit dem Zusatz ».chk«. Diese können Sie später zum Beipiel mit dem CLI-Befehl TYPE ohne erneute Berechnung der Prüfsumme noch einmal ausgeben.

Haben Sie »Nur Checksumme ausgegeben?« mit »n« beantwortet, dann können Sie dem Programm den Vergleich überlassen, in dem die Frage »Eingabe aus Datei« mit »j« beantwortet wird. Dann brauchen Sie nur noch die Checksummen eingeben. Der Checksummer holt sich die Zeile aus der angegebenen Datei statt von der Tastatur. Entspricht die eingegebene Prüfzahl nicht der errechneten, kann die Zeile gleich korrigiert werden.

Beantworten Sie obige Frage mit »n«, zählt Checkie die in der Datei vorhandenen Zeilen und wartet mit der Zeilennummer »Anzahl+1« auf die Eingabe einer neuen Zeile. Alle weiteren Eingaben hängt das Programm an die bestehende Datei an. Diese Funktion ist sinnvoll, wenn Sie ein Listing in mehreren Teilen abtippen wollen.

So tippen Sie Listings ab

Haben Sie anfangs einen Dateinamen eingegeben oder Sie hängen neue Zeilen an eine bestehende Datei an, dann arbeiten Sie im normalen Eingabemodus. Checkie 42 schlägt dabei eine Zeilennummer vor und wartet auf die Prüfsumme. Nach Eingabe derselben taucht der Cursor zwischen den zwei Trennstrichen auf. Dort muß nun die Zeile »ohne« Zeilennumer und Prüfsumme eingegeben werden. Nach Betätigen der Taste <Return> berechnet Checkie die Prüfsumme. Leerstellen vor und hinter der Programmanweisung werden ignoriert. Stimmen Programmzeile und Prüfsumme mit derjenigen im Listing überein, speichert der Checksummer die Eingabe ab und wartet auf die nächste Zeile.

Einfügemodus: Wahrscheinlich wird eine abgetippte Zeile mal einen Fehler enthalten. Checkie 42 positioniert den Cursor dann an den Anfang der Zeile und wartet auf die korrekte Eingabe. Korrekturen lassen sich mit der Backspace- oder Delete-Taste durchführen. Um Zeichenfolgen einzufügen, kann kurzfristig mit <F2> der Einfügemodus eingeschaltet werden. Dieser Modus sollte allerdings nach der Fehlerkorrektur wieder ausgeschaltet werden, da er die Eingabe verlangsamt.

Sonderfall-Prüfsumme ignorieren: Möchten Sie zum Beispiel eine Kommentarzeile nicht »original« übernehmen, läßt sich trotz einer falschen Prüfsumme eine Übernahme der Zeile mit der Funktionstaste <F6> erzwingen. Sie können damit aber auch falsche Programmzeilen übernehmen. Verwenden Sie deshalb die Taste <F6> nicht gewohnheitsmäßig. Der Checksummer teilt Ihnen nach Beenden des Programms mit, wieviel Zeilen er ungeprüft übernommen hat.

Prüfsumme und Zeilennummer ändern: Natürlich kann es auch vorkommen, daß die Programmzeile zwar richtig abgetippt wurde, sich bei der Prüfsumme aber ein Fehler eingeschlichen hat. Nach Betätigen von <F1> kann die Prüfsumme korrigiert werden. Während der Eingabe der Prüfsumme läßt sich mit <F7> die vom Programm vorgeschlagene Zeilennummer verändern. Damit können Sie gezielt nur bestimmte Teile eines Listings übernehmen.

Haben Sie eine mit einem anderen Editor geschriebene Programmdatei überprüft und nur in wenigen Zeilen Fehler festgestellt, lassen sich durch Vorgabe der Nummern diese Zeilen gezielt ändern. Bei Angabe der Zeilennummer in aufsteigender Reihenfolge benötigt das Programm übrigens erheblich weniger Zeit für die Suche der Zeilen in der jeweiligen Datei. Um die versehentliche Übernahme fehlerhafter Zeilen zu verhindern, sperrt das Programm bei fehlender Übereinstimmung der Prüfsummen die Taste <F7> (Änderung der Zeilennummer).

Fehlerfrei abtippen

Eingabe beenden: Die Kombination <Ctrl-e> beendet den Programmlauf nach vollständiger Eingabe des Listings oder für eine Unterbrechung.

Am Schluß noch ein Tip für alle Leser, denen unser Basic-Editor zu langsam ist. Die Berechnung der Prüfsummen erfolgt im Unterprogramm »CalcSumme«. Dieser Teil ist sehr einfach in schnellere Sprachen, wie beispielsweise C, umsetzbar.

Wer schon einmal Fehler in einem abgetippten Listing gesucht hat, der weiß, wie frustrierend diese Arbeit sein kann. Nutzen Sie deshalb den »Checkie 42«. Sie sparen viel Zeit und müssen sich nicht dauernd auf die Suche nach tückischen Fehlern begeben.

(Dieter Behlich/kn)

```
CqO REM ***********************************
KF  REM *** Checksummer: Checkie 42       ***
4c  REM ***          Version 1.1          ***
Ft  REM ***********************************
Z7  Start:
W42   GOSUB Init
Pn    GOSUB OpenDatei
mb    IF dn$="" THEN Ende
b2    GOSUB Bild
OwO NeueZeile:
K42   GOSUB loeschen
zT    GOSUB EingabeSumme
pt    IF FEnde=wahr THEN Ende
MVO Wiederholung:
sQ2   GOSUB EingabeZeile
sw    IF FEnde=wahr THEN Ende
mO    GOSUB CalcSumme
fB    IF FZok = falsch THEN GOTO Wiederholung
oC    GOSUB Uebernahme
iv    GOTO NeueZeile
mAO Ende:
7S2   GOSUB fertig
72    END
UGO Init:
fD2   wahr=-1
fy    falsch=0
Kh    LZeile=240  : REM Anzahl Zeichen/Zeile
tz    LBZeile=60  : REM Anzahl Zeichen/Bildschirmzeile
OG    AnzBZeilen=LZeile/LBZeile
Nl    zx=6 : zy=14   : REM Position Zeile
PC    sx=20 : sy=5   : REM Position Checksumme
mF    AnzCsZ=3       : REM Anzahl Ziffern/Checksumme
QF    DIM z(LZeile)
05    DIM cs(AnzCsZ*2)
iz    cs(AnzCsZ)=0   : REM Zeilenstart
gv    a=0 : b=0 : c=0  : REM Hilfsvariablen
AI    i=0 : j=0 : k=0  : REM Zählvariablen
XL    FZok = wahr : FCz3=100
Wn    Checkfile=0  : Zeile=1
Bb    READ Faktor(i)
X1    WHILE Faktor(i)<>0
ga4     i=i+1
cm      READ Faktor(i)  : REM Faktorenreihe
9x2   WEND
Gm    AnzFak=i
RB    DATA 2,3,4,5,6,0
7j    RETURN
PWO OpenDatei:
BR2   ON ERROR GOTO Dateifehler
MS    CLS
Vu    LOCATE 3,5:dn$="":INPUT "Dateiname: ";dn$
Fp    OPEN dn$ FOR INPUT AS #1
KU    IF FFehler=53 THEN NeueDatei
4s    LOCATE 5,5:PRINT "Nur Checksummen ausgeben? (j/n) ":e$=INP
      UT$(1)
gM    IF e$="j" THEN
Kq4     LOCATE 7,5:e$="":PRINT "Bildschirm oder Drucker? (b/d) "
        :e$=INPUT$(1)
Ku      IF e$="d" THEN
KM7       OPEN "prt:" FOR OUTPUT AS #4
YH4     ELSE
7Y6       OPEN dn$+".chk" FOR OUTPUT AS #4
vo4     END IF
RR      WHILE NOT EOF(1)
yJ6       LINE INPUT#1,e$
fg        apos=LEN(e$)
to        z(apos)=32
4L        FOR i=1 TO apos
sq8         z(i-1)=ASC(MID$(e$,i,1))
Xn6       NEXT i
cE        GOSUB CalcSumme
vp        i=0
Rh        WHILE z(i)=32 : i=i+1 : WEND
4v        IF FCz3=i THEN
f98         cs(6)=-16    : REM Leerstelle
nW6       ELSE
XA8         cs(6)=i : FCz3=i
A36       END IF
2o        PRINT USING "####";Zeile; : PRINT " ";
Wf        PRINT#4,USING "####";Zeile; : PRINT#4," ";
mC        FOR i=4 TO 6
7d8         IF cs(i)>35 THEN cs(i)=cs(i)+6
nS          IF cs(i)>9 THEN cs(i)=cs(i)+7
jy          PRINT CHR$(cs(i)+48);
8L          PRINT#4,CHR$(cs(i)+48);
n36       NEXT i
19        PRINT " ";e$
tl        PRINT#4," ";e$
Qm        Zeile=Zeile+1
rf4     WEND
IF      CLOSE 1 : CLOSE 4
H4      dn$=""
Fv      PRINT:PRINT "Taste drücken"
5r      e$=INPUT$(1)
rT      RETURN
SL2   END IF
70    LOCATE 7,5:PRINT"Eingabe aus Datei? (j/n) ":e$=INPUT$(1)
L1    IF e$="j" THEN
u24     Checkfile=1
08      CLOSE 1
eX      GOSUB backup
WZ      OPEN dn$+".bak" FOR INPUT AS #1
Sk      OPEN dn$ FOR OUTPUT AS #2
Oc      RETURN
bU2   END IF
77    WHILE NOT EOF(1)
eP4     LINE INPUT#1,e$
j5      Zeile=Zeile+1
Ay2   WEND
YI    CLOSE 1
8AO NeueDatei:
aM2   OPEN dn$ FOR APPEND AS #2
91    RETURN
7QO Dateifehler:
Og2   FFehler=ERR
PB    IF FFehler=53 THEN RESUME NEXT
vy    ON ERROR GOTO 0
aDO backup:
yY2   OPEN dn$ FOR INPUT AS#1
Sj    OPEN dn$+".bak" FOR OUTPUT AS #2
8F    flen=LOF(1)
ih    WHILE flen>=32000
i14     e$=INPUT$(32000,1)
NK      PRINT#2,e$;:flen=flen-32000
QE2   WEND
e7    IF flen THEN
SB4     e$=INPUT$(flen,1)
xP      PRINT#2,e$;
zs2   END IF
zg    CLOSE 2  : CLOSE 1
R3    RETURN
dOO Bild:
f12   CLS
Gi    LOCATE 2,10
yY    PRINT ">>>>>>> Checkie 42 - Der ultimative Checksum
      mer! <<<<<<<"
xR    LOCATE 13,zx
Le    FOR i=1 TO 2
kx4     FOR j=1 TO LBZeile
HR6       PRINT "-";
hy4     NEXT j
H1      LOCATE 13+AnzBZeilen+1,zx
hx2   NEXT i
Hi    LOCATE 5,30:PRINT "<CTRL e> = Programm beenden"
vd    LOCATE 7,8:PRINT "<F2> = Einfügemodus ein"
uQ    LOCATE 7,40:PRINT "<F1> = Checksumme ändern"
1H    LOCATE 9,40:PRINT "<F6> = Zeile speichern"
ME    LOCATE 9,8:PRINT "<F3> = Zeile löschen"
v5    LOCATE 11,8:PRINT "<F7> = Zeilennummer ändern"
jL    RETURN
5fO loeschen:
w82   LOCATE zy,1
8S    FOR i=1 TO AnzBZeilen
jf4     PRINT TAB(zx);SPACE$(LBZeile)
t92   NEXT i
OO    FOR i=0 TO LZeile
V84     z(i)=32
wC2   NEXT i
3F    LOCATE zy,1
om    PRINT USING "####";Zeile
dy    apos=0
vX    RETURN
v5O EingabeSumme:
va2   Farbe=0 : i=1
AB    LOCATE sy,sx+1
Xm    PRINT "..."
O4O blinken:
aC2   Farbe=Farbe XOR 1
QH    COLOR Farbe,0
P6    LOCATE sy,sx-15
1n    PRINT "Checksumme:"
ni    LOCATE sy,sx+i
ya    e$=INKEY$
lT    IF e$="" THEN blinken
Xp    COLOR 1,0
dl    e=ASC(e$)
fM    IF e=5 THEN
KU4     FEnde=wahr   : REM <CTRL e>
hf      i=AnzCsZ+1   : REM fertig
Cc2   ELSEIF e=135 THEN
dc4     GOSUB NeuZeile
Bz      e=135
```

```
LB2   ELSEIF e=8 THEN
4h4     IF i>1 THEN
sZ6       i=i-1       : REM <BS>
XY        LOCATE sy,sx+i : PRINT "."
un4     END IF
Tn2   ELSEIF e=13 THEN
KZ4     IF i=AnzCsZ THEN i=AnzCsZ+1 : REM <CR>
cL2   ELSE
ra4     IF e>47 AND e<58 THEN
vi6       e=e-48                    : REM 0-9
AI4     ELSEIF e>64 AND e<91 THEN
Rv6       e=e-55                    : REM A-Z
jX4     ELSEIF e>96 AND e<123 THEN
Zx6       e=e-61                    : REM a-z
jS4     ELSE
yz6       GOTO blinken              : REM weder noch
6z4     END IF
JY      PRINT e$;
ve      cs(i)=e
D7      i=i+1
A32   END IF
wn    IF i<=AnzCsZ THEN blinken
f50 ESEnde:
2K2   COLOR 1,0
Oh    LOCATE sy,sx-15
cO    PRINT "Checksumme:"
gI    RETURN
aEO NeuZeile:
dU2   IF FZok = wahr THEN
P44     NeuZeile=0
nn      WHILE e<>13 OR NeuZeile=0
Ux6       LOCATE zy,1:PRINT USING "####";NeuZeile;
oF        e=ASC(INPUT$(1))
gP        IF e>47 AND e<58 THEN NeuZeile=NeuZeile*10+e-48
Jn        IF NeuZeile > 9999 THEN e=8
in        IF e=8 THEN NeuZeile=INT(NeuZeile/10)
vj4     WEND
kh      IF Checkfile THEN
tQ6       IF NeuZeile < Zeile THEN
yy8         WHILE NOT EOF(1)
VGA           LINE INPUT#1,e$
cd            PRINT#2,e$
1p8         WEND
MH          CLOSE 1 : CLOSE 2
fY          GOSUB backup
Xa          OPEN dn$+".bak" FOR INPUT AS #1
Tl          OPEN dn$ FOR OUTPUT AS #2
9t          Zeile=1
cV6       END IF
Yp        WHILE (NeuZeile > Zeile) AND (NOT EOF(1))
fQ8         LINE INPUT#1,e$
mn          PRINT#2,e$
17          Zeile=Zeile+1
CO6       WEND
cR        IF EOF(1) THEN
bL8         CLOSE 1
8i          NeuZeile=Zeile
uN          LOCATE zy,1:PRINT USING "####";NeuZeile;
9G          Checkfile=0
ng6       END IF
oh4     END IF
vV      Zeile=NeuZeile
qj2   END IF
HtO RETURN
1H  EingabeZeile:
n72   x=cs(AnzCsZ)
BWO weiter:
ga2   cy=zy+INT(x/LBZeile):cx=zx+(x MOD LBZeile)
vy    LOCATE cy,cx
o6    COLOR 0,1
mI    PRINT CHR$(z(x));
y1    LOCATE cy,cx
68    IF x>apos THEN apos=x
xx    IF Checkfile AND FZok THEN
w14     IF EOF(1) THEN
OR6       Checkfile=0 : CLOSE 1
jS4     ELSE
ix6       e$=INPUT$(1,1)
6z4     END IF
mV2   ELSE
Q24     e$=INKEY$
922   END IF
gp4     IF e$="" THEN weiter
OI      COLOR 1,0
OW      PRINT CHR$(z(x));
CF      LOCATE cy,cx
8G      e=ASC(e$)
UR2   IF ((e AND 127)<32) OR e=127 THEN Controlcode
op    IF imode THEN GOSUB insert
I6    PRINT e$
Ki    z(x)=e : e=30
3DO Controlcode:
gF2   IF e=13 OR e=10 THEN
1N4     RETURN
092   ELSEIF e=30 THEN
2p4     a=1
Ea2   ELSEIF e=29 THEN
034     a=LBZeile
6G2   ELSEIF e=31 THEN
DD4     a=-1
Gb2   ELSEIF e=28 THEN
Rg4     a=-LBZeile
9s2   ELSE
UX4     GOTO noCrs
WP2   END IF
hz    x=x+a
5Z    IF x>=0 AND x<LZeile THEN weiter
tD    x=x-a
HB    GOTO weiter
kEO noCrs:
dN2   IF e=8 THEN
1s4     IF x>0 THEN
Kk6       x=x-1
AM        LOCATE zy+INT(x/LBZeile),zx+(x MOD LBZeile)
Bb        FOR i=x TO apos
788         z(i)=z(i+1)
Ja          PRINT CHR$(z(i));
ZO          IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
FV6       NEXT i
pr        z(apos)=32 : PRINT " "
yR        apos=apos-1
ng4     END IF
Hi2   ELSEIF e=127 THEN
Kk4     FOR i=x TO apos
GH6       z(i)=z(i+1)
Sj        PRINT CHR$(z(i));
i9        IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
Oe4     NEXT i
yO      z(apos)=32 : PRINT " "
7a      apos=apos-1
Vy2   ELSEIF e=129 THEN
oI4     GOSUB EingabeSumme
sC      x=cs(AnzCsZ)
9U2   ELSEIF e=130 THEN
513    imode=imode XOR 1
tU5      LOCATE 7,28
yZ3    IF imode THEN
iT5      PRINT "aus"
jS3    ELSE
YV5      PRINT "ein"
6z3    END IF
Kg2   ELSEIF e=131 THEN
L54     GOSUB loeschen
3N      x=cs(AnzCsZ)
7Y2   ELSEIF e=134  THEN
bD4     RETURN
PR2   ELSEIF e=5 THEN
ZT4     FEnde=wahr
eG      RETURN
F82   END IF
xr    GOTO weiter
IfO insert:
oJ2   IF apos>x THEN
3o4     FOR i=apos TO x STEP -1
Qt6       z(i+1)=z(i)
q64     NEXT i
PH      z(x)=32
Ny      apos=apos+1
Kp      IF apos=LZeile THEN apos=apos-1:z(LZeile)=32
uK      FOR i=x TO apos
1I6       PRINT CHR$(z(i));
Hi        IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
xD4     NEXT i
yA      LOCATE zy+INT(x/LBZeile),zx+(x MOD LBZeile)
UN2   END IF
vX    RETURN
cXO CalcSumme:
3s2   a=0 : b=0 : c=0
z2    IF e=134 THEN
pv4     FZok=wahr
WX      FF6=FF6+1
Gz2   ELSE
hv4     WHILE z(apos)=32 AND apos>0
pI6       apos=apos-1
9x4     WEND
Ni      IF apos>0 THEN
036       WHILE z(c)=32
```

Listing. Der verbesserte »Checkie 42«. Bitte mit der ersten Version von »Checkie 42« oder ohne Prüfsummen und Zeilennummern eingeben.

EINGABEHILFE

```
Bt8          c=c+1
D16        WEND
jc4      END IF
EJ       FOR i=c TO apos
Y26        j=(i-c) MOD AnzFak
Qs         k=(i+1-c) MOD AnzFak
s1         a=a+((z(i) AND 127)-32)*Faktor(j)
2E         b=b+((z(i) AND 127)-32)*Faktor(k)
Ka4      NEXT i
pA       cs(4)=a+Zeile-INT((a+Zeile)/62)*62
4N       cs(5)=b+Zeile-INT((b+Zeile)/62)*62
4Q       FZok=(cs(1)=cs(4)) AND (cs(2)=cs(5))
tm2    END IF
Kw     RETURN
pMO Uebernahme:
xD2    FOR i=0 TO apos
ih4      PRINT#2,CHR$(z(i));
Tj2    NEXT i
VZ4      PRINT#2,""
5R2    Zeile=Zeile+1
R3     RETURN
N90 fertig:
MJ2    IF Checkfile THEN
ZZ4      WHILE NOT EOF(1)
6r6        LINE INPUT#1,e$
DE         PRINT#2,e$
cQ4      WEND
Ok       CLOSE 1
922    END IF
5q     CLOSE 2
nt     CLS
Rg     LOCATE 12,35
WL     PRINT "F E R T I G !!!"
I9     LOCATE 20,1
D1     IF FF6<>0 THEN
C14      PRINT "ACHTUNG!!!  ";
9H       PRINT FF6;" Zeile(n) wurde(n) ungeprüft gespeichert."
IB2    END IF
jL     RETURN
(C) 1988 M&T
```

Listing. Der verbesserte »Checkie 42« (Schluß)

Impressum

Herausgeber: *Carl-Franz von Quadt, Otmar Weber*

Chefredakteur: *Hans-Günther Beer*
stellv. Chefredakteur: *Gottfried Knechtel – verantwortlich für den redaktionellen Teil*
Chef vom Dienst: *Susanne Kirmaier*
Redaktion: *Ralf Sablowski, Klaus Sonnenleiter*
Redaktionsassistenz: *Brigitte Bobenstetter, Sylvia Derenthal, Helga Weber (202)*
Mitarbeiter der Redaktion: *Martin Jobst, Andreas Lietz, Christian Wolf, Heiko Schlichting, Peter Wollschläger, Johann Waldhäusel, Arno Gölzer, Nikolaus Huber, Jürgen Siegbert*

Alle Artikel sind mit dem Kennzeichen des Redakteurs (kn = Gottfried Knechtel, rs = Ralf Sablowski, so = Klaus Sonnenleiter) und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Art-director: *Friedemann Porscha*
Layout: *Erich Schulze (Cheflayouter), Rudolf Weikl, Marian Schwarz, Andrea Miller*
Fotografie: *Sabine Tennstaedt, Ilona Wiewiorra*
Titelgestaltung: *Friedemann Porscha*
Spritzgrafik: *Norbert Raab*
Computergrafik: *Werner Nienstedt*

Auslandsrepräsentation:
Schweiz: Markt&Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug, Tel. 042-415656, Telex: 862329 mut ch
USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063, Telefon: (415) 366-3600, Telex 752-351
Österreich: Markt&Technik Ges. mbH Hermann Raniger, Große Neugasse 28, A 1040-Wien, Tel. 0043-222-8579455, Telex: 047-132532

Manuskripteinsendungen: *Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen und dazu, daß Markt&Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.*

Produktionsleiter: *Klaus Buck (180)*

Anzeigenleitung: *Alicia Clees (313) – verantwortlich für Anzeigen*

Anzeigenformate: *1/1 Seite ist 266 Millimeter hoch und 185 Millimeter breit (2 Spalten à 86 Millimeter oder 4 Spalten à 43 Millimeter). Vollformat 297x210 Millimeter.*

Anzeigenverwaltung und Disposition: *Lisa Landthaler (233)*

Anzeigen-Auslandsvertretung: England: *F. A. Smyth&Associates Limited, 23a, Aylmer Parade, London, N2 OPQ. Telefon:0044/1/3405058, Telefax: 0044/1/3419602*
Taiwan: *Third Wave Publishing Corp., 1-4 Fl. 977 Min Shen E. Road, Taipei 10581, Taiwan, R.O.C., Tel. 00886/2/7630052, Telefax: 00886/2/7658767, Telex: 078529335*

Vertriebsleiter: *Helmut Grünfeldt (189)*

Verkaufsleiter Abonnement: *Benno Gaab (740)*

Verkaufsleiter Einzelhandel: *Robert Riesinger (364)*

Vertrieb Handelsauflage: *Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1,*

Bezugsmöglichkeiten: *Leser-Service: Telefon (089) 4613-366. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.*

Preis: *Das Einzelheft kostet DM 16,–*

Druck: *SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg*

Sonderdruck-Dienst: *Alle in dieser Ausgabe erschienenen Beiträge sind in Form von Sonderdrucken zu erhalten. Anfragen an Reinhard Jarczok, Tel. 089/4613-185, Fax 4613-776.*

Redaktionsdirektor: *Michael M. Pauly*

Vorstand: *Otmar Weber (Vors.), Bernd Balzer*

Leiter Unternehmensbereich »Populäre Computerzeitschriften«:
Eduard Heilmayr, Werner Pest

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: *Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0, Telex 5-22052*

Telefon-Durchwahl im Verlag: *Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089/4613 und dann die Nummer, die in den Klammern hinter dem jeweiligen Namen angegeben ist.*

Mitteilung gemäß Bayerischem Pressegesetz: *Aktionäre, die mehr als 25% des Kapitals halten: Otmar Weber, Ingenieur, München; Carl-Franz von Quadt, Betriebswirt, München;*
Aufsichtsrat: *Carl-Franz von Quadt (Vorsitzender), Dr. Robert Dissmann (stellv. Vorsitzender), Eduard Heilmayr*

Textverarbeitung par excellence

Textverarbeitungsprogramme auf dem Amiga wurden bisher von Profis belächelt. Sie waren langsam wie Schreibmaschinen, bunt wie Kindergeburtstage, absturzsicher wie Starfighter oder teuer wie Schweizer Uhren. Bis Documentum kam… Rasend schnell ● Automatische Trennhilfe ● Fußnotenverwaltung ● Editier-/WYSIWYG-Bildschirm ● Blockbearbeitung ● Kopf-/Fußzeilen ● Seitennumerierung ● Blockbearbeitung ● Alle Amiga-Zeichensätze nutzbar ● Tastatur- oder Mausbedienung ● Buchdruckoption ● Suchen/Ersetzen ● NLQ-Druck für alle Drucker ● multitaskingfähig ● und vieles, vieles mehr…

Für Briefschreiber, Studierende, Autoren, Bürokraten – eigentlich für alle.

Bestell-Nr. 54122

DM 149,–* (sFr 135,–*/öS 1490,–*)

* Unverbindliche Preisempfehlung



220901

Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computer-Fachgeschäften oder bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0; Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526; Ueberreuter Media Verlagsges.mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.



Die Welt von Disneyland – jetzt auf Ihrem Amiga. Mit **MovieSetter** erstellen Sie Trickfilme – auch wenn Sie kein Profi sind. MovieSetter ermöglicht es, in kürzester Zeit komplexe, minutenlange Animationssequenzen zu erstellen – mit insgesamt nur 1 Mbyte Speicher. Per Mausklick erzeugen Sie aus vorher erstellten Brushes eine fließende Bewegung vor feststehendem oder beweglichem Hintergrund. Plazieren Sie Geräusche innerhalb des Programms und verändern Sie die Tondauer, Tonlage und -stärke; durch Steuern der Tonkanäle können Sie auch Stereoeffekte erzielen.

Zahlreiche vorgefertigte Movie-Clips werden mitgeliefert. Spezielle Animationseffekte erreichen Sie durch Farbdurchlauf oder Playback von bis zu 60 Sequenzen pro Sekunde.
Dieses neue Animationsprogramm ist nicht nur für den Anfänger leicht zu erlernen, es ist auch für den Trickfilmprofi ein vielseitiges Werkzeug.

Bestell-Nr.: 54128, Preis: DM 198,–* (sFr 178,–*/öS 1980,–*)
Deutsche Version in Vorbereitung.
Update DM 49,–* (sFr 49,–*/öS 490,–*)
(lieferbar 1. Quartal 1989).

*Unverbindliche Preisempfehlung

Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computer-Fachgeschäften oder bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0; Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526; Ueberreuter Media Verlagsges.mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.

225-2/811